Università degli studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di laurea Magistrale in Ingegneria Informatica

Riconoscimento di facce in un sistema di visione distribuito

Face recognition in a distributed vision system

Laureando: Carlo Giuliani

Relatore: Prof. Emanuele Menegatti

Correlatore: dott. Salvatore M. Anzalone

Anno Accademico 2010/2011

Alla mia famiglia, che mi ha sempre sostenuto.

A mia madre, che avrebbe tanto desiderato
vivere con me questo traguardo.

E a tutte quelle persone che hanno contribuito
a rendere unici questi 5 meravigliosi anni.

Indice

Sommario

Questa tesi è stata svolta nell'ambito di un progetto presso lo IAS-Lab (Laboratorio di Sistemi Autonomi Intelligenti) dell'Università degli Studi di Padova. Il progetto consiste nella creazione di un sistema autonomo di videosorveglianza distribuito.
La tesi ha come obiettivo principale la creazione di un sistema di rilevamento e riconoscimento delle facce distribuito in una rete di sensori; in particolare si è voluto creare un sistema in grado di elaborare flussi video provenienti da sensori installati in posizioni diverse all'interno di una rete locale, allo scopo di estrarre da essi le immagini relative alle facce delle persone presenti all'interno di una stanza e inviarle successivamente ad un server centrale, incaricato di effettuarne il riconoscimento. Lo scopo principale di tale sistema è quello di cercare di riconoscere eventuali persone all'interno della stanza anche qualora esse non siano visibili pienamente da alcuni dei sensori presenti, sfruttando in questo modo la distribuzione delle varie telecamere all'interno dell'ambiente.
Questa tesi di laurea si basa principalmente su due aspetti:

- la creazione di un modello di riferimento della stanza in cui si trovano i sensori comune a tutti questi ultimi;
- l'utilizzo di un middleware, chiamato NMM, per la creazione di una rete di nodi finalizzata alla distribuzione del software di rilevamento delle persone e al riconoscimento di queste ultime.

In questa tesi si farà uso dell'algoritmo di Viola-Jones per effettuare il rilevamento delle persone e delle facce e dell'algoritmo Eigenfaces per il riconoscimento di queste ultime; verranno inoltre analizzate le prestazioni dell'algoritmo di riconoscimento e proposti dei miglioramenti volti ad aumentare le prestazioni complessive del sistema.

Abstract

This thesis has been carried out within a project at IAS-Lab (Intelligent Autonomous Systems Lab), University of Padua. The project aims to create an autonomous system for distributed video surveillance.

The thesis had as main objective the creation of a face detection and recognition system in a distributed sensors network. In particular, we wanted to create a system capable of processing video streams from sensors installed in different locations within a local network, in order to extract from them the images of the people faces in a room and then send the selected images to a central server, instructed to carry out the recognition. The main purpose of this system is to recognize people in the room even if they are not fully visible from some sensors, thus exploiting the distribution of the various cameras in the room.

This thesis is mainly based on two aspects:

- the creation of a reference model of the room in which the sensors are common to all of them;
- the use of a middleware, called NMM, to create a network of nodes which distributes people and face detection and face recognition within a Local Area Network.

In this thesis we will use the Viola-Jones algorithm to perform the detection of people and faces and the Eigenfaces algorithm for the recognition of the latter; we will also analyze the performance of the algorithm for recognition and propose some improvements aimed to increase the overall performance of the system.

1 Introduzione

L'obiettivo principale dell'*ambient intelligence* è quello di distribuire l'intelligenza tra i vari agenti e oggetti che compongono un ambiente. Tipicamente tali sistemi sono basati su una rete distribuita di sensori e attuatori sparsi nell'ambiente che cercano di catturare informazioni rilevanti e che possono agire all'interno dell'ambiente (1) (2). Questo approccio implica sia la memorizzazione delle informazioni che il loro utilizzo in tempo reale, al fine di portare a termine i compiti assegnati, come consentire l'accesso alle persone autorizzate o inviare segnalazioni qualora si verifichino eventi non previsti. Un aspetto importante in questo senso è l'estrazione delle informazioni di interesse dai flussi di dati provenienti dai vari sensori: questi ultimi, generalmente, contengono una grande quantità di informazione necessaria, e a volte l'informazione di interesse è "nascosta" all'interno di essa. In questo senso, potrebbe non essere un problema banale per una persona umana, soggetta a diversi livelli di stress e attenzione, identificare tali informazioni all'interno dei flussi video provenienti dai sensori.

Nel mondo moderno, sempre più connesso alla rete, il bisogno di mantenere la sicurezza delle informazioni sta diventando sempre più importante e allo stesso tempo di difficile gestione. Si sente sempre più spesso parlare di atti criminali legati al furto di carte di credito, all'intrusione all'interno delle reti informatiche da parte di hacker e all'accesso non autorizzato all'interno di aziende o edifici governativi. Nella maggior parte di queste situazioni, i criminali vengono favoriti dal principale difetto dei sistemi di controllo d'accesso attuali: l'accesso viene garantito non da chi si è, ma in base a cosa si possiede, come badge, chiavi, password o PIN. Nessuno di questi metodi serve a definire chi siamo; essi sono piuttosto metodi per l'autenticazione. Tuttavia, è sufficiente rubare una di tali autenticazione per avere accesso ai dai personali di una persona. Si è quindi sempre più alla ricerca di tecnologie che siano in grado di riconoscere la vera identità di una persona. In questo senso, i sistemi di controllo d'accesso biometrici sono metodi automatici per verificare o riconoscere l'identità di una persona sulla base di alcune sue caratteristiche fisiche, come le impronte digitali o la sua faccia. Tali sistemi, basandosi su caratteristiche biologiche dell'individuo, sono più difficili da aggirare rispetto ai comuni metodi di autenticazione.
Il riconoscimento di facce automatico è un concetto relativamente nuovo. Sviluppato negli anni 60, il primo sistema semi-automatico per il riconoscimento delle facce richiedeva all'amministratore di localizzare manualmente caratteristiche come naso, capelli, occhi e bocca all'interno delle fotografie; si procedeva quindi a calcolare le distanze e i rapporti tra queste

ultime e un punto di riferimento comune, per confrontare infine tali risultati con i dati di riferimento.

Nei primi anni 70, vennero introdotti 21 indicatori soggettivi, come il colore dei capelli o lo spessore delle labbra, per automatizzare il processo di riconoscimento (3). Tuttavia, entrambi questi primi approcci, presentavano il problema di dover calcolare manualmente le misure e le posizioni delle features.

Nel 1988, si applicò per la prima volta l'analisi delle componenti principali al problema del riconoscimento delle facce (4). Tale applicazione dimostrò come fossero necessari meno di cento valori per codificare correttamente l'immagine di una faccia. Nel 1991, Turk e Pentland proposero un metodo, chiamato Eigenfaces, con cui si dimostrò la possibilità di realizzare sistemi di riconoscimento delle facce che operino in tempo reale (5).

Attualmente, la tecnologia per il riconoscimento delle facce viene utilizzata per combattere il furto di passaporti, supportare le forze dell'ordine, identificare i bambini scomparsi e minimizzare i furti di identità.

Tra le varie tecniche biometriche attualmente presenti il riconoscimento delle facce non è sicuramente la più affidabile ed efficiente, tuttavia presenta numerosi vantaggi rispetto alle altre: è naturale, facile da utilizzare e non richiede aiuti da parte del soggetto da riconoscere. Per questi motivi, sistemi di riconoscimento delle facce installati ad esempio negli aeroporti possono rilevare la presenza di criminali anche in mezzo ad una folla; altre tecnologie biometriche come le impronte digitali o il riconoscimento tramite la voce non sono in grado di effettuare tali scansioni di massa.

Tuttavia, nonostante il successo di molti sistemi di riconoscimento delle facce, restano ancora parecchi aspetti che dovranno essere soggetti ad ulteriori ricerche. Tra questi, vanno citati i problemi legati all'illuminazione, alla posa, alla scalatura e ad immagini prese a distanza di anni tra loro.

La rilevazione delle facce è il primo passo per il riconoscimento delle stesse: essa consiste nell'identificare quali parti dell'immagine contengono la faccia di una persona.

L'intero processo di riconoscimento delle facce può quindi idealmente essere suddiviso in due fasi: la rilevazione e il riconoscimento vero e proprio. Tale suddivisione è rappresentata in Figura 1.1:

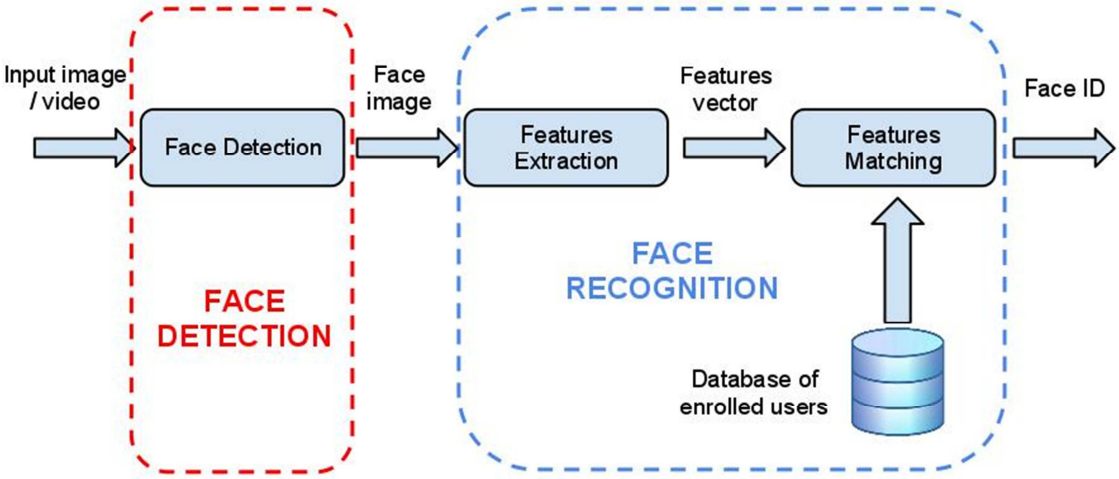


Figura 1.1 - Fasi dell'attività di riconoscimento delle facce.

1.1 Ambiente e strumenti di lavoro

Questo lavoro di tesi è stato svolto all'interno dello IASLab (Laboratorio di Sistemi Autonomi Intelligenti) dell'Università di Padova, presso il dipartimento di Ingegneria dell'Informazione.

In particolare, è stata utilizzata la rete di sensori presente all'interno del laboratorio, attualmente consistente in tre telecamere, di cui due fisse e una telecamera PTZ. In particolare, le due telecamere fisse sono due telecamere Panasonic WVCP240EX, mentre la telecamera PTZ utilizzata è una Ulisse Camera, prodotta dall'azienda VideoTec S.p.A. La Figura 1.2 mostra le due tipologie di telecamere utilizzate:

(a) **(b)**

Figura 1.2 - Le due tipologie di telecamere utilizzate.

La Figura 1.3 e la Figura 1.4 mostrano rispettivamente il laboratorio dove è stato svolto questo lavoro di tesi e la posizione delle telecamere all'interno dello stesso:

Figura 1.3 - Due immagini dello IASLab

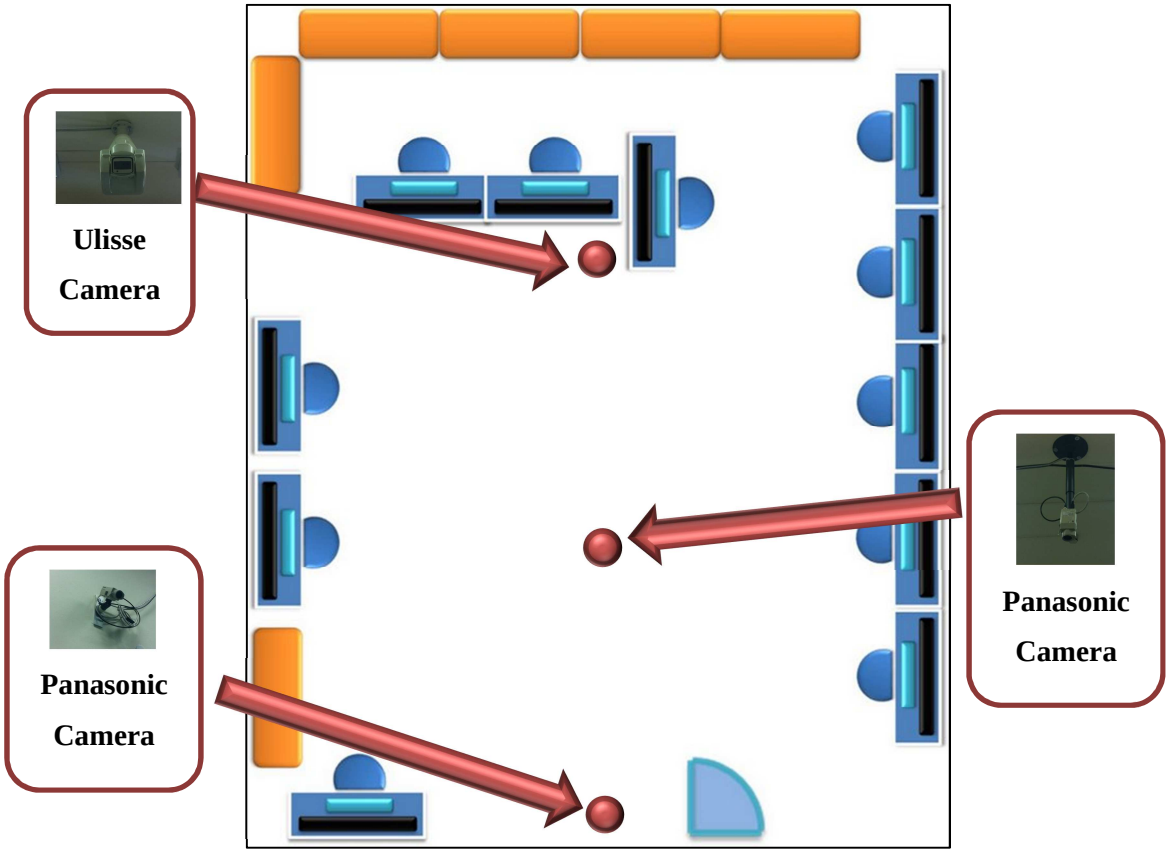


Figura 1.4 - Posizione delle telecamere all'interno del laboratorio

Questo lavoro di tesi si inserisce all'interno di un progetto più ampio, *IDVS* (Intelligent Distributed Video Systems for Surveillance & Quality Inspection), finanziato dalla Regione Veneto (Progetto 2105/201/4/1268/2008) e volto a creare una procedura per la fusione intelligente dei dati provenienti da vari sensori, creare un sistema di cooperazione autonoma tra di essi e rilevare il verificarsi di eventi anomali.

1.2 Scopo della tesi

Questa tesi di laurea ha come obiettivo quello di creare un'applicazione che, dati vari flussi video provenienti dalle diverse telecamere presenti nel sistema, sia in grado di rilevare le

persone all'interno della stanza e successivamente di riconoscerle. In particolare, si vuole porre l'accento sul fatto che l'applicazione agisce in una rete di sensori; inoltre tali sensori non sono necessariamente collegati alla stessa macchina ma possono trovarsi in posizioni diverse all'interno della rete locale.

Il lavoro di tesi svolto può quindi idealmente essere suddiviso in tre fasi:

- La creazione di un modello di riferimento comune a tutta la rete di sensori attraverso la calibrazione delle telecamere e la realizzazione di funzioni che consentono di passare dalle immagini prese dalle varie telecamere al sistema di riferimento scelto e viceversa;
- L'integrazione di tale modello con il software già presente per il rilevamento di persone e facce all'interno di un'immagine;
- La creazione di un nodo che, ricevendo in ingresso i dati dai vari sensori presenti all'interno della rete locale LAN, sia in grado di "fondere" insieme tali dati e successivamente di applicare il riconoscimento delle facce agli stessi.

1.3 Stato dell'arte

Vi sono diversi approcci al problema del riconoscimento delle facce. Uno dei più famosi è sicuramente l'algoritmo noto come Eigenfaces, introdotto nel 1991 dai ricercatori Turk e Pentland (5). Questo algoritmo fa uso dell'analisi delle componenti principali (PCA) per ridurre la dimensione dei dati da analizzare, rimuovendo l'informazione non necessaria e decomponendo la struttura della faccia in componenti ortogonali non correlate tra loro.

Un altro approccio, chiamato Fisherfaces, è stato presentato nel 1997 dai ricercatori Belhumeur, Hespanha e Kriegman della Yale University (6). L'obiettivo di questi ultimi era quello di trovare una proiezione lineare delle facce dallo spazio delle immagini, caratterizzato da un numero molto elevato di dimensioni, ad uno spazio delle features con poche dimensioni, e che fosse allo stesso tempo poco sensibile alle variazioni di illuminazione e espressività delle facce. L'algoritmo FisherFaces fa uso del metodo noto come *Fisher's Linear Discriminant* (FLD) (7), che tenta di massimizzare la dispersione tra classi diverse all'interno dello spazio delle features e allo stesso tempo minimizzare quella relativa a immagini appartenenti alla stessa classe.

Nel 2009 i ricercatori Dreuw, Steingrube, Hanselmann e Ney delle Aachen University hanno proposto un approccio che si basa sulle features SURF (Speeded Up Robust Features), features invarianti rispetto alla rotazione, scalatura e illuminazione (8) (9).

Tutte le tecniche fin qui esposte fanno parte del riconoscimento di facce a due dimensioni. Nel 2005 i ricercatori A. Bronstein, M. Bronstein e R. Kimmel hanno introdotto il riconoscimento delle facce in tre dimensioni (10), tecnica che, assumendo che le espressioni facciali possano essere modellate come isometrie sulla superficie della faccia, consentiva di costruire delle

rappresentazioni delle facce invarianti rispetto alle differenti espressioni delle stesse. Le tecniche di riconoscimento delle facce 3D promettono inoltre di essere più resistenti rispetto alle variazioni delle condizioni di illuminazione e di posa delle varie immagini, tuttavia sono ancora oggetto di grande ricerca; inoltre gli hardware di acquisizione 3D sono attualmente ancora molto costosi e la sostituzione dei vecchi apparati 2D con quest'ultimi è un processo che richiederà comunque dei tempi abbastanza lunghi.
In questo lavoro di tesi si è fatto uso dell'approccio Eigenfaces, in quanto l'obiettivo principale non era quello di creare un riconoscitore che avesse delle ottime perfomance, bensì quello di individuare un algoritmo che consentisse la distribuzione dello stesso; l'algoritmo Eigenfaces si è rilevato essere la scelta più opportuna in quanto già implementato all'interno delle librerie opencv.

Recentemente vi sono stati sviluppi anche in merito alla distribuzione dell'algoritmo per il riconoscimento delle facce. Roberto Tron e René Vidal hanno proposto un algoritmo, chiamato *CONSENSUS*, in cui le varie telecamere presenti nel sistema, dopo aver riconosciuto una persona, si scambiano tali risultati in modo da ottenere un consenso globale sulla sua identità (11). Girija Chetty e Dharmendra Sharma, dell'Università di Canberra, hanno proposto un'applicazione multi-agente al problema del riconoscimento di facce, in grado di ottenere più del 95% di accuracy anche in caso di variazioni delle condizioni di illuminazione, posa ed espressioni della faccia (12). Xie, Boult, Ramesh e Zhu, dell'Università del Colorado, hanno proposto un sistema di riconoscimento delle facce che utilizza più di una telecamera: essi definiscono una misura di affidabilità che viene associata al risultato del riconoscimento effettuato da ciascuna telecamera, scegliendo poi come risultato finale quello avente la misura di affidabilità più elevata (13). Simile approccio è stato utilizzato dai ricercatori Harguess, Hu e Aggarwal, dell'Università del Texas, che utilizzano più di una telecamera assegnando vari pesi al risultato del riconoscimento effettuato da queste ultime; tali pesi sono determinati in base all'orientazione della faccia nell'immagine (14).
L'algoritmo utilizzato all'interno di questo lavoro di tesi, al contrario dei precedenti, fa uso si un sistema di riconoscimento centralizzato e basa la fusione dei dati provenienti dalle diverse telecamere sulla posizione all'interno della stanza della persona rilevata dalle stesse: tale posizione può essere determinata grazie al fatto che tutte le telecamere presenti nel sistema sono calibrate. In questo modo è possibile scambiare informazioni riguardo al riconoscimento delle varie persone tra le diverse telecamere: ad esempio una telecamera, sebbene veda una persona di schiena, può ricevere informazioni circa la sua identità da un'altra telecamera che la vede di fronte e che è quindi in grado di effettuare un riconoscimento sulla sua faccia.

Il lavoro di tesi svolto si basa sulla calibrazione delle varie telecamere presenti nel sistema. Per quanto riguarda la calibrazione delle telecamere fisse, va citato l'approccio ormai standard esposto anche nel libro *"Learning OpenCV"* (15) e di cui esistono molte implementazioni, tra cui quella presente nelle librerie OpenCV stesse. Per quanto riguarda la calibrazione delle telecamere PTZ, si può citare il lavoro dei ricercatori Riera, Salvador e Casas, dell'Università di Barcellona, che hanno sviluppato un metodo in grado di calibrare una telecamera PTZ tenendo in considerazione anche il fatto che essa può variare i suoi angoli di pan e di tilt (16).
Tuttavia tale implementazione assume che gli assi di rotazione della telecamera PTZ siano paralleli agli assi del sistema di riferimento della telecamera quando essa viene calibrata; in questo lavoro di tesi si è quindi dovuto studiare un metodo alternativo, in quanto tale assunzione, nel caso della telecamera PTZ presente nel laboratorio, non è verificata.

1.4 Struttura della tesi

Questo lavoro di tesi è così articolato: nel capitolo 2 verranno introdotti gli algoritmi utilizzati per creare un modello di riferimento comune a tutte le telecamere presenti nel sistema, che tengono in considerazione anche la presenza di telecamere PTZ.
Successivamente, nel capitolo 3 verrà introdotto NMM, il middleware utilizzato per la distribuzione dei flussi video e dei dati all'interno della rete locale, ponendo l'accento anche sulla procedura di installazione del software e sulle sue attuali limitazioni;
Verrà poi introdotto nel capitolo 4 l'algoritmo utilizzato per la rilevazione delle persone e delle immagini all'interno del flusso video proveniente dalla telecamera, e verrà inoltre presentata l'implementazione realizzata all'interno di NMM.
Il capitolo 5 tratta dell'algoritmo utilizzato per il riconoscimento delle facce e dei vari test effettuati per verificare le performance dello stesso mentre Il capitolo 6 tratta del nodo realizzato all'interno di NMM per gestire la distribuzione dell'applicazione.
Infine, nel capitolo 7 verranno esposte le conclusioni di questo lavoro di tesi e i possibili sviluppi futuri.

2 Modelizzazione 3D dell'ambiente

In questo capitolo verrà presentata la procedura e l'algoritmo utilizzato per creare un modello 3D dell'ambiente comune a tutta la rete di telecamere. In particolare, verranno affrontati i seguenti punti:

- Calibrazione delle telecamere al fine di ottenere i parametri estrinseci (matrice di rotazione e vettore di traslazione) rispetto ad un sistema di riferimento comune.
- Realizzazione delle funzioni cam2world e word2cam per le telecamere fisse.
- Realizzazione delle funzioni cam2world e world2cam per la telecamera PTZ Ulisse
- Realizzazione di una funzione che, dato un punto 3D espresso nelle coordinate del sistema di riferimento mondo, sia in grado di calcolare gli angoli di pan e tilt con cui far ruotare la telecamera per centrare il punto nell'immagine.

L'obiettivo ultimo di questa parte della tesi è stato quindi la creazione di un modulo che permettesse l'utilizzo di un sistema di riferimento comune tra tutte le telecamere, e quindi, ad esempio, l'utilizzo di tale sistema per scambiare nella rete di telecamere le coordinate di qualsiasi tipo di oggetto visto da queste ultime.

2.1 Calibrazione delle telecamere

Per prima cosa è stato necessario calibrare ciascuna telecamera del sistema. La calibrazione di ciascuna telecamera è stata suddivisa in due fasi, la prima finalizzata a ricavare i parametri intrinseci della telecamera, e la seconda a ricavare la matrice di rototraslazione rispetto ad un sistema di riferimento comune. In particolare, i parametri intrinseci di una telecamera sono i parametri necessari a collegare le coordinate di un pixel dell'immagine con le coordinate 3D corrispondenti nel sistema di riferimento della telecamera; i parametri estrinseci sono i parametri che definiscono la posizione e l'orientazione del sistema di riferimento della telecamera rispetto al sistema di riferimento mondo, che è supposto essere noto.

2.1.1 Modello pinhole e calcolo dei parametri intrinseci

La calibrazione effettuata si basa sul modello *pinhole* della telecamera. In questo modello si assume che la luce entri nel sistema dalla scena o da un oggetto posto ad una certa distanza, ma per ciascun punto entra nel sistema un singolo raggio di luce. Tale punto viene quindi proiettato in una superficie, detta *piano immagine*. Come risultato, l'immagine nel piano immagine è sempre a fuoco, e le dimensioni dell'immagine relative ad un oggetto posto ad una certa distanza dipendono da un solo parametro, chiamato *lunghezza focale*. Nel modello ideale di una telecamera pinhole, la lunghezza focale corrisponde esattamente alla distanza tra il piano

immagine e l'apertura della telecamera. Tale modello è rappresentato dalla Figura 2.1 in cui *f* è la lunghezza focale della telecamera, *Z* è la distanza della telecamera dall'oggetto, *X* è la lunghezza dell'oggetto e *x* è l'immagine dell'oggetto nel piano immagine.

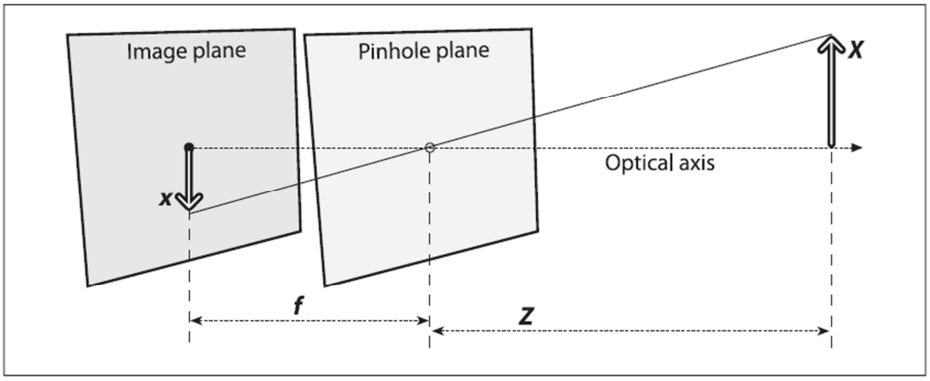


Figura 2.1 - Modello pinhole: un foro (detto foro di apertura) lascia passare solo quei raggi che intersecano un particolare punto nello spazio; tali raggi vengono proiettati nel piano immagine formando l'immagine (Immagine presa dal libro "Gary Bradski & Adrian Kaehler, Learning OpenCV - Computer Vision with the OpenCV Library, O'REILLY (15)).

Questo modello può essere modificato al fine di ottenerne uno equivalente, ma nel quale i calcoli risultino semplificati. Nella Figura 2.2 sono stati invertiti il foro di apertura e il piano immagine. Il punto che corrisponde al foro di apertura viene chiamato *centro della proiezione*, e corrisponde all'origine degli assi del sistema di riferimento della telecamera. In tale modello, ogni raggio parte da un oggetto posto ad una certa distanza e termina nel centro della proiezione. Il punto che corrisponde all'intersezione tra il piano immagine e l'asse ottico della telecamera è detto *punto principale* (ed è uno dei parametri intrinseci restituiti dalla calibrazione). L'immagine viene generata intersecando i raggi con il piano immagine, esattamente a distanza *f* dal centro della proiezione.

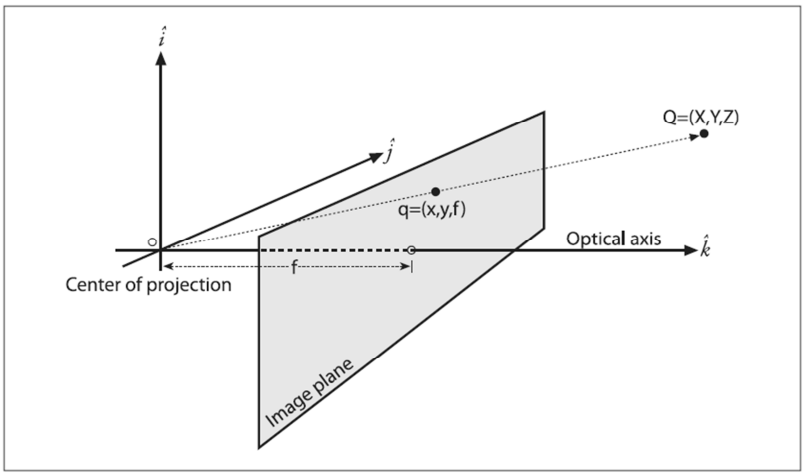


Figura 2.2 - Un punto $Q = (X, Y, Z)$ viene proiettato nel piano immagine attraverso il raggio che lo congiunge con il centro della proiezione, ottenendo il punto nell'immagine $q = (x, y, f)$ (Immagine presa dal libro "Gary Bradski & Adrian Kaehler, Learning OpenCV - Computer Vision with the OpenCV Library, O'REILLY (15)).

È importante notare come il punto principale in generale non corrisponda al centro dell'immagine: solitamente infatti il centro dell'immagine non si trova mai esattamente sull'asse ottico della telecamera. È pertanto necessario introdurre due nuovi parametri, c_x e c_y, che modellino lo scostamento del centro dell'immagine dall'asse ottico della telecamera.
Il risultato di tale modello è che un punto Q nel mondo avente coordinate (X, Y, Z) viene proiettato nel piano immagine in un punto avente coordinate x e y, date dalle seguenti equazioni:

$$x = f_x\left(\frac{X}{Z}\right) + c_x$$

$$y = f_y\left(\frac{Y}{Z}\right) + c_y$$

Si noti infine come nelle precedenti equazioni siano state introdotte due lunghezze focali: ciò è dovuto al fatto che nelle telecamere di fascia medio – bassa i pixel sono rettangolari anziché quadrati.
La relazione che mappa un punto Q_i nel mondo avente coordinate (X_i, Y_i, Z_i) nei punti sul piano immagine aventi coordinate (x_i, y_i) è detta *trasformazione proiettiva*. Per semplificare i calcoli, è conveniente rappresentare i punti in *coordinate omogenee*. Le coordinate omogenee associate ad un punto in uno spazio a *n* dimensioni sono rappresentate da un vettore avente *n+1* dimensioni (ad esempio *x*, *y*, *z* diventano *x*, *y*, *z*, *w*), con la restrizione aggiuntiva che due punti i cui valori sono proporzionali sono equivalenti.
Nel caso di una telecamera, i punti sono rappresentati da coordinate bidimensionali, pertanto un punto nell'immagine in coordinate omogenee viene rappresentato da un vettore tridimensionale $q = (q_1, q_2, q_3)$. Poiché tutti i punti aventi valori proporzionali nello spazio proiettivo sono equivalenti, è possibile ricavare le coordinate in pixel di un punto dividendo le coordinate omogenee per q_3. Ciò consente anche di rappresentare i parametri intrinseci che definiscono la telecamera (f_x, f_y, c_x, c_y) all'interno di un'unica matrice 3x3, detta *matrice dei parametri intrinseci della telecamera*. La proiezione di un punto 3D del mondo espresso nel sistema di riferimento della telecamera in un punto nel piano immagine può essere quindi riassunta dalla seguente equazione:

$$q = MQ \qquad \text{dove} \qquad q = \begin{bmatrix} x \\ y \\ w \end{bmatrix}, M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Il modello esposto fino ad adesso non tiene conto della distorsione della lente. Poiché nella realtà nessuna lente è perfetta, verrà sempre introdotta una distorsione nell'immagine.
Per ottenere la matrice con i parametri intrinseci della telecamera, sono state scattate per ciascuna di esse un numero di fotografie ad una scacchiera, posizionata in diverse orientazioni

e a diverse distanze. Successivamente tali fotografie sono state elaborate tramite delle funzioni presenti nella libreria opencv, ottenendo al termine la matrice dei parametri intrinseci.

2.1.2 Calcolo dei parametri estrinseci

Per ogni immagine di un particolare oggetto ripreso dalla telecamera, è possibile descrivere la posizione dell'oggetto rispetto al sistema di riferimento della telecamera attraverso una rotazione ed una traslazione, come rappresentato in Figura 2.3:

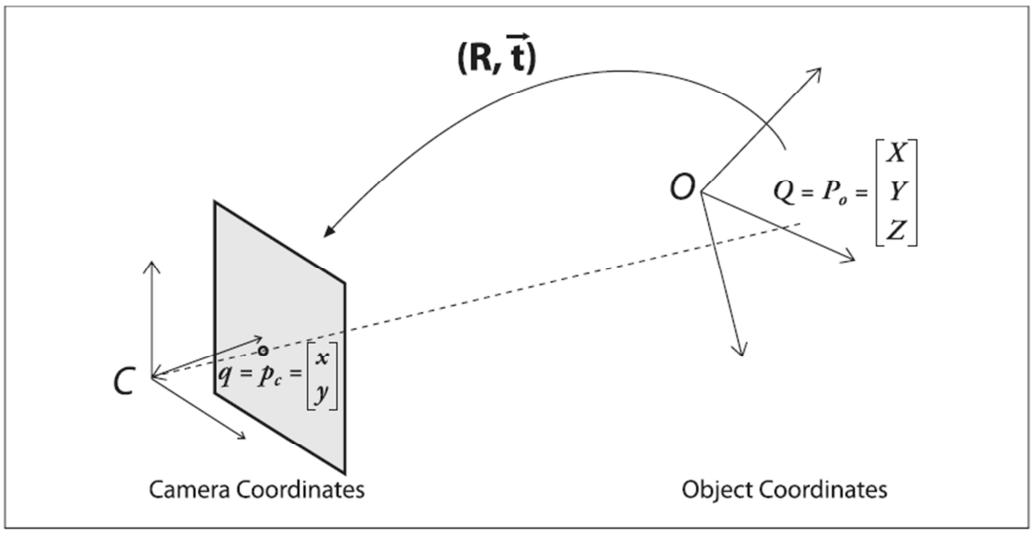


Figura 2.3 - Conversione dal sistema di riferimento dell'oggetto al sistema di riferimento della telecamera: il punto P_0 dell'oggetto è visto come il punto p_c nel piano immagine; il punto p_c è ottenuto dal punto P_0 moltiplicando quest'ultimo per la matrice di rotazione R e per vettore di traslazione t (Immagine presa dal libro "Gary Bradski & Adrian Kaehler, Learning OpenCV - Computer Vision with the OpenCV Library, O'REILLY (15)).

In generale, una rotazione in un qualsiasi numero di dimensioni può essere descritta come la moltiplicazione di un vettore di coordinate per una matrice di dimensioni opportune. La rotazione corrisponde ad introdurre una nuova descrizione della posizione del punto in un diverso sistema di riferimento. La rappresentazione di una rotazione bidimensionale come la moltiplicazione tra matrici è rappresentata in Figura 2.4:

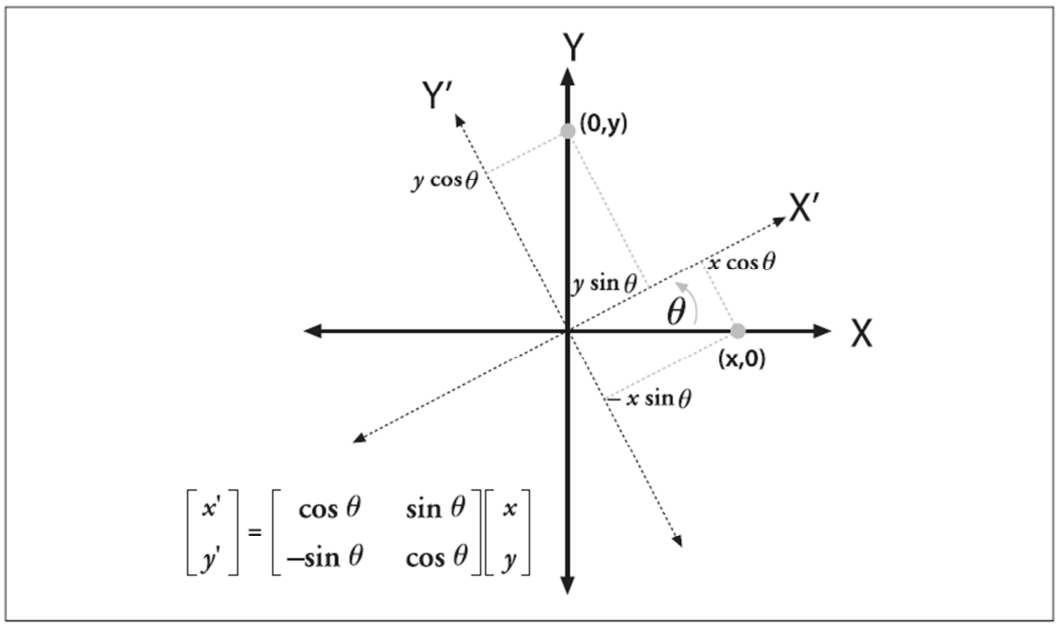


Figura 2.4 - Rotazione bidimensionale di θ rispetto all'origine degli assi (Immagine presa dal libro "Gary Bradski & Adrian Kaehler, Learning OpenCV - Computer Vision with the OpenCV Library, O'REILLY (15)).

La rotazione in tre dimensioni può essere scomposta in rotazioni bidimensionali attorno a ciascuno degli assi. La rotazione in sequenza attorno agli assi *x*, *y*, *z* con angoli di rotazione ψ, φ, θ è rappresentata dalla matrice di rotazione R ottenuta dal prodotto delle matrici $R_x(\psi), R_y(\varphi), R_z(\theta)$ dove

$$R_x(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\psi) & \sin(\psi) \\ 0 & -\sin(\psi) & \cos(\psi) \end{bmatrix}$$

$$R_y(\varphi) = \begin{bmatrix} \cos(\varphi) & 0 & -\sin(\varphi) \\ 0 & 1 & 0 \\ \sin(\varphi) & 0 & \cos(\varphi) \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

e $R = R_x(\psi)R_y(\varphi)R_z(\theta)$.

Il vettore di traslazione rappresenta lo scostamento da un sistema di riferimento ad un altro i cui assi hanno la stessa orientazione del primo. Detto in un altro modo, il vettore di traslazione rappresenta semplicemente l'offset dall'origine del primo sistema di riferimento all'origine del secondo.

Per ricavare la matrice di rotazione e il vettore di traslazione della telecamera rispetto al sistema di riferimento del mondo, è stata posta una scacchiera sul piano pavimento della stanza in una posizione nota. Fornendo le coordinate nel sistema mondo di tale scacchiera ad un programma appositamente creato, attraverso l'uso della funzione

`FindExtrinsicCameraParams2` presente nella libreria OpenCV, è stato possibile ricavare la matrice di rotazione ed il vettore di traslazione per ciascuna delle telecamere utilizzate.

Dalla matrice di rotazione $R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ e dal vettore di traslazione $T = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$ è possibile ottenere la matrice di rototraslazione complessiva, rappresentata attraverso una matrice 4x4:

$$A = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Questa matrice consente di passare dal sistema mondo al sistema di riferimento della telecamera; per passare dal sistema di riferimento della telecamera al sistema di riferimento mondo è sufficiente calcolare la matrice inversa A^{-1}.

Le matrici di rototraslazione consentono di ricavare anche la posizione delle telecamere e dei loro sistemi di riferimento rispetto al sistema di riferimento mondo, illustrate nella Figura 2.5:

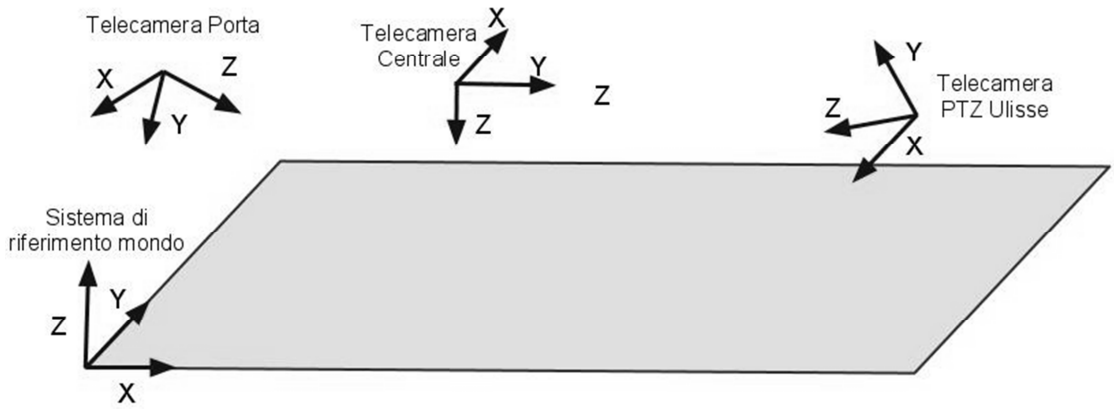


Figura 2.5 - Posizione dei sistemi di riferimento delle telecamere rispetto al sistema di riferimento mondo.

2.2 Realizzazione delle funzioni `cam2world` e `world2cam` per le telecamere fisse

Il passo successivo è stato l'implementazione delle funzioni `world2cam` e `cam2world` per le telecamere fisse: la prima consente di ottenere le coordinate in pixel nell'immagine della telecamera di un punto 3D espresso con coordinate nel sistema di riferimento mondo, mentre la seconda consente di effettuare l'operazione inversa, limitatamente ai punti che si trovano in un piano prestabilito (in questo caso il piano rappresentato dal pavimento della stanza).

2.2.1 `world2cam`

La realizzazione della funzione `world2cam` per una telecamera fissa, in base a quanto esposto nei paragrafi precedenti, non è altro che l'implementazione della seguente equazione:

$$s\begin{bmatrix}u\\v\\1\end{bmatrix}=\begin{bmatrix}f_x & 0 & c_x\\0 & f_y & c_y\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}r_{11} & r_{12} & r_{13} & t_1\\r_{21} & r_{22} & r_{23} & t_2\\r_{31} & r_{32} & r_{33} & t_3\end{bmatrix}\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix}$$

in cui (u, v) sono le coordinate del punto in pixel nell'immagine, (X, Y, Z) sono le coordinate del punto nel sistema di riferimento mondo, $\begin{bmatrix}f_x & 0 & c_x\\0 & f_y & c_y\\0 & 0 & 1\end{bmatrix}$ è la matrice dei parametri intrinseci della telecamera e $\begin{bmatrix}r_{11} & r_{12} & r_{13} & t_1\\r_{21} & r_{22} & r_{23} & t_2\\r_{31} & r_{32} & r_{33} & t_3\end{bmatrix}$ è la matrice di rototraslazione.

2.2.2 cam2world

La realizzazione della funzione `cam2world` si è rivelata essere più complessa della funzione `world2cam`. Innanzitutto, poiché le telecamere sono state calibrate singolarmente bisogna assumere che tutti i punti dell'immagine di cui si vogliono le coordinate 3D nel sistema mondo appartengano ad uno stesso piano. Infatti nel modello pinhole descritto nei paragrafi precedenti tutti i punti appartenenti ad una retta che attraversa il foro di apertura della telecamera vengono mappati in un unico punto dell'immagine; con le informazioni provenienti dalla calibrazione della singola telecamera è impossibile capire quale di questi punti appartenenti alla retta sia effettivamente il punto di interesse. Nell'implementazione realizzata si è quindi assunto che tutti i punti ottenuti come output dalla funzione `cam2world` appartengano al piano descritto dal pavimento. Il fine ultimo di tale applicazione è infatti il calcolo nel sistema di riferimento mondo delle coordinate in cui si trova una persona: poiché quest'ultima deve appoggiare i piedi sul pavimento, non è sbagliato assumere che tutti i punti di cui si vogliono conoscere le coordinate 3D appartengano a tale piano.

L'algoritmo implementato ricava l'equazione della retta passante per centro della proiezione e punto nell'immagine e l'equazione del piano che identifica il pavimento: l'intersezione tra retta e piano è il punto richiesto. Vediamo ora nel dettaglio l'algoritmo implementato, di cui si fornisce lo pseudocodice:

```
Function cam2world(x, y, M, A)

Input: (x, y) coordinate del punto in pixel
Input: M matrice dei parametri intrinseci della telecamera
Input: A matrice di rototraslazione della telecamera rispetto al
       sistema di riferimento mondo
Output: (X,Y,Z) coordinate 3D del punto nel sistema di riferimento
        mondo (con Z=0)

(new_x, new_y) ← coordinate in pixel del punto (x,y) rispetto ad un
             sistema di riferimento centrato in (M.cx, M.cy).
```

```
f ← (M.f_x + M.f_y) / 2
point1 ← (0,0,0)
point2 ← (1,0,0)
point3 ← (0,1,0)
point1_cam ← rototranslate_point(point1, A)
point2_cam ← rototranslate_point(point2, A)
point3_cam ← rototranslate_point(point3, A)
vect ← vettore normale al piano identificato dai punti point1_cam,
              point2_cam, point3_cam
I ← punto di intersezione tra il piano identificato da vect e la retta
    passante per i punti (0,0,0) e (new_x,new_y,f)
I_world ← rototranslate_point(I, A^-1)
return I_world
```

Figura 2.6 - Pseudocodice della funzione `cam2world` implementata

Per capire meglio il funzionamento dell'algoritmo descritto dal precedente pseudocodice si faccia riferimento alla Figura 2.7:

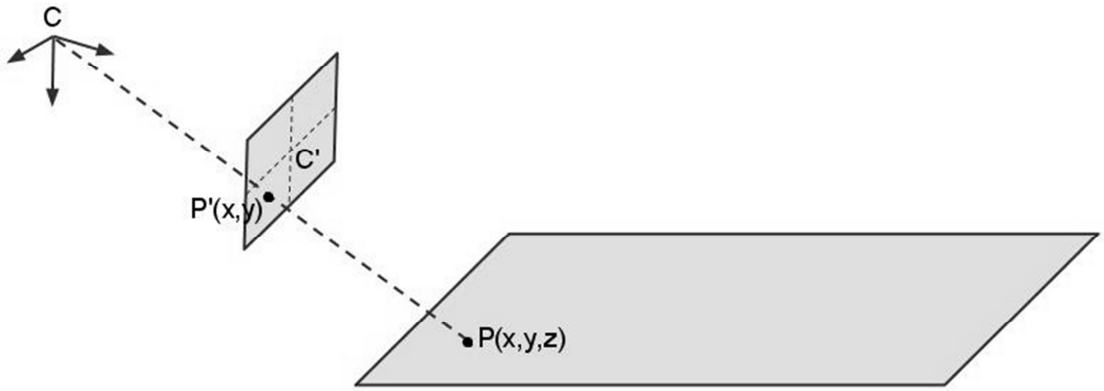


Figura 2.7 - Rappresentazione grafica dell'algoritmo cam2world

Per prima cosa, l'algoritmo cambia il sistema di riferimento dell'immagine della telecamera in un sistema di riferimento in cui gli assi hanno la stessa orientazione, ma in cui l'origine di questi ultimi coincide con il punto principale. Fatto questo, l'algoritmo calcola la coordinata z del punto in pixel rispetto al sistema di riferimento della telecamera: tale coordinata corrisponde semplicemente alla lunghezza focale della telecamera stessa.

Successivamente l'algoritmo calcola il vettore normale al piano pavimento, che identifica in maniera univoca quest'ultimo. Poiché ogni piano è identificato univocamente da tre punti, e il piano pavimento è composto da tutti i punti del sistema di riferimento mondo aventi la terza coordinata pari a zero, è sufficiente scegliere tre di questi punti, in questo caso (0,0,0) (1,0,0) e (0,1,0). Poiché tuttavia i calcoli risultano più semplici se effettuati nel sistema di riferimento

della telecamera, l'algoritmo effettua una rototraslazione di questi punti per ottenere le coordinate corrispondenti nel sistema di riferimento della telecamera e successivamente utilizza tali nuove coordinate per calcolare il vettore normale al piano pavimento: in questo modo è come se si disponesse dell'equazione del piano del pavimento nel sistema di riferimento della telecamera.

In tale sistema di riferimento, il centro della proiezione ha coordinate (0,0,0): l'algoritmo calcola quindi l'intersezione tra la retta passante per questo punto e per il punto in pixel in coordinate 3D e il piano del pavimento nel sistema di riferimento della telecamera. A questo punto si dispone del punto desiderato rispetto al sistema di riferimento della telecamera: è sufficiente quindi applicare una rototraslazione inversa per ottenere il punto in coordinate del sistema di riferimento mondo.

L'algoritmo implementato utilizza due funzioni ausiliarie, la prima che calcola il vettore normale ad un piano identificato da tre punti e la seconda che calcola l'intersezione tra un piano e una retta passante per due punti, di cui si riporta il codice MATLAB (tali funzioni sono state riscritte in linguaggio C++, tuttavia il codice MATLAB è di più facile lettura).

```
function vec = planenormvec(p1,p2,p3)

% PLANENORMVEC Calculate the normal vector of a plane in 3D.
%
%   VEC = PLANENORMVEC(PT1,PT2,PT3) calculats the normal
%   vector of a plane that includes three points PT1, PT2,
%   PT3. When the plane is expressed as ax + by + cz = 1,
%   VEC(1) = a, VEC(2) = b, VEC(3) = c.
%   Points should be a 1 x 3 vector, specifying x, y and z
%   values for each column.

A = [p1;p2;p3];
if cond(A) ~= inf % when not singular
    d = [1;1;1];
    vec = inv(A)*d;
else % when it passes [0,0,0]
    B = [p1(1),p1(2);p2(1),p2(2)];
    if cond(B) == inf
        error('Three points are badly placed.');
        return %break
    end
    d = [-p1(3);-p2(3)];
```

```
    foo = inv(B)*d;
    vec = [foo(1);foo(2);1];
end
```

Figura 2.8 - Codice MATLAB della funzione ausiliaria per il calcolo del vettore normale al piano

```
function [I,check]=plane_line_intersect(n,V0,P0,P1)
%plane_line_intersect computes the intersection of a plane and a
segment(or
%a straight line)
% Inputs:
%       n: normal vector of the Plane
%       V0: any point that belongs to the Plane
%       P0: end point 1 of the segment P0P1
%       P1:  end point 2 of the segment P0P1
%
%Outputs:
%      I    is the point of interection
%     Check is an indicator:
%      0 => disjoint (no intersection)
%      1 => the plane intersects P0P1 in the unique point I
%      2 => the segment lies in the plane
%      3=>the intersection lies outside the segment P0P1
%
I=[0 0 0];
u = P1-P0;
w = P0 - V0;
D = dot(n,u);
N = -dot(n,w);
check=0;
if abs(D) < 10^-7        % The segment is parallel to plane
        if N == 0           % The segment lies in plane
            check=2;
            return
        else
            check=0;       %no intersection
            return
        end
end
```

```
%compute the intersection parameter
sI = N / D;
I = P0+ sI.*u;


if (sI < 0 || sI > 1)
    check= 3;              %The intersection point  lies outside the
segment, so there is no intersection
else
    check=1;
end
```

Figura 2.9 - Codice MATLAB della funzione ausiliaria per il calcolo dell'intersezione tra un piano e una retta.

La funzione `planenormvec` è stata scritta da Yo Fukushima[1], mentre la funzione `plane_line_intersect` da Nassim Khaled, della Wayne State University.

2.2.3 Test degli algoritmi `cam2world` e `world2cam`

Successivamente, è stato effettuato un test delle funzioni `cam2world` e `world2cam` descritte nei paragrafi 2.2.1 e 2.2.2 utilizzando le due telecamere fisse disponibili, situate rispettivamente sopra la porta d'ingresso del laboratorio e al centro dello stesso. In particolare, il test consisteva nell'inserire in input le coordinate espresse nel sistema di riferimento globale di punti sul pavimento, e verificare di quanto si discostavano da quelle reali le coordinate in pixel in uscita dalla funzione `cam2world`. Successivamente, sono state inserite in ingresso le coordinate in pixel di punti noti sul pavimento della stanza al fine di verificare l'errore introdotto dalla funzione `world2cam`.

I risultati di tali test sono riassunti nelle seguenti tabelle:

`cam2world` Telecamera fissa sopra la porta d'ingresso								
Coordinate globali (mm)			**Pixel restituiti**		**Pixel effettivi**		**Errore (pixel)**	
x	**y**	**z**	**x**	**y**	**x**	**y**	**x**	**y**
3000	2000	0	380	313	387	307	7	6
4000	2000	0	365	254	373	249	8	5
4000	4000	0	205	254	216	251	11	3

Tabella 2.1 - Test della funzione `cam2world` sulla telecamera fissa posizionata sopra la porta d'ingresso del laboratorio.

[1] http://www.mathworks.com/matlabcentral/fileexchange/authors/17527

world2cam Telecamera fissa sopra la porta d'ingresso									
Pixel di input		**Coordinate di output (mm)**			**Coordinate reali (mm)**			**Errore (mm)**	
x	**y**	**x**	**y**	**z**	**x**	**y**	**z**	**x**	**y**
387	307	3097	1907	0	3000	2000	0	97	93
373	249	4120	1890	0	4000	2000	0	120	110
216	251	4079	3882	0	4000	4000	0	79	118

Tabella 2.2 - Test della funzione world2cam sulla telecamera fissa situata sopra la porta d'ingresso del laboratorio.

cam2world Telecamera fissa al centro del laboratorio								
Coordinate globali (mm)			**Pixel restituiti**		**Pixel effettivi**		**Errore (pixel)**	
x	**y**	**z**	**x**	**y**	**x**	**y**	**x**	**y**
3000	2000	0	64	166	69	164	5	2
4000	2000	0	73	374	83	365	10	9
4000	4000	0	484	349	490	342	6	7

Tabella 2.3 - Test della funzione cam2world sulla telecamera fissa situata al centro del laboratorio.

world2cam Telecamera fissa al centro del laboratorio									
Pixel di input		**Coordinate di output (mm)**			**Coordinate reali (mm)**			**Errore (mm)**	
x	**y**	**x**	**y**	**z**	**x**	**y**	**z**	**x**	**y**
69	164	2988	2029	0	3000	2000	0	12	29
83	365	3960	2055	0	4000	2000	0	40	55
490	342	3966	4024	0	4000	4000	0	34	24

Tabella 2.4 - Test della funzione world2cam sulla telecamera fissa situata al centro del laboratorio.

Dall'analisi di queste tabelle, si può osservare come, nel caso della telecamera posta sopra la porta d'ingresso, la funzione cam2world introduca un errore massimo pari all'1,72% della risoluzione totale (usando immagini con risoluzione 640x480), mentre la funzione world2cam introduce un errore massimo pari a 12 cm. Nel caso della telecamera situata al centro del laboratorio si ha un errore inferiore, pari all'1,56% della risoluzione totale per quanto riguarda la funzione cam2world e a 5,5 cm per quando riguarda la funzione world2cam.

La Figura 2.10 e la Figura 2.11 illustrano la differenza tra i punti calcolati e i punti effettivi per quanto riguarda la funzione cam2world:

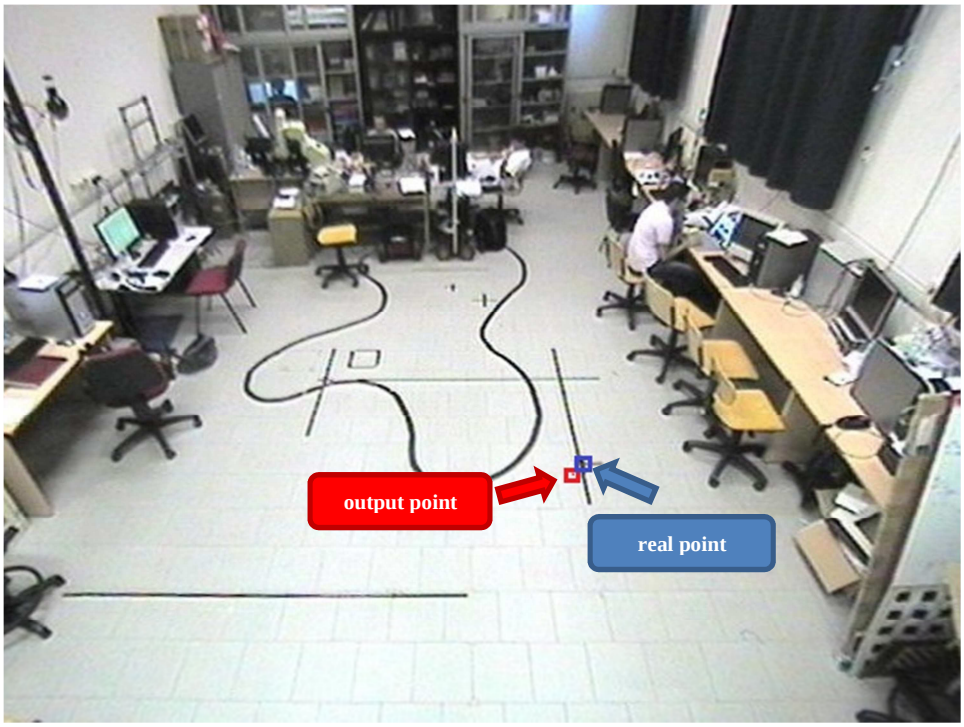


Figura 2.10 - Esempio di errore della funzione world2cam applicata alla telecamera fissa situata sopra la porta d'ingresso del laboratorio.

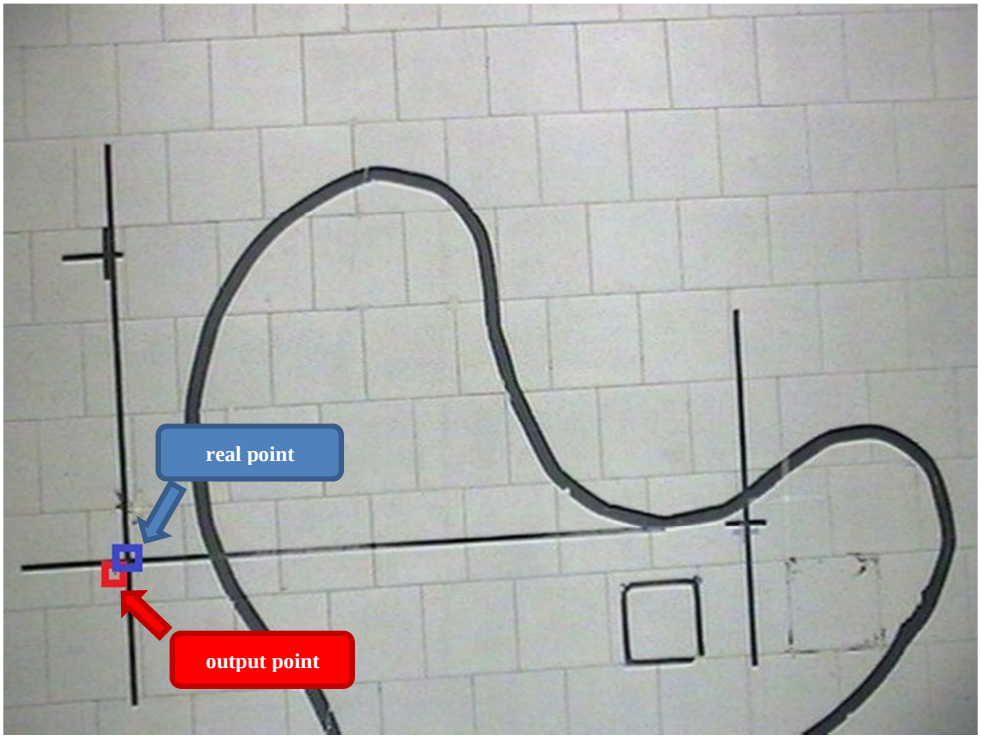


Figura 2.11 - Esempio di errore della funzione world2cam applicata alla telecamera fissa situata sopra la porta d'ingresso del laboratorio.

2.3 Realizzazione delle funzioni `world2cam` e `cam2world` per la telecamera PTZ Ulisse

Le funzioni descritte nei paragrafi 2.2.1 e 2.2.2 assumono che la telecamera sia sempre fissa rispetto alla posizione di calibrazione, e non si adattano quindi ad una telecamera PTZ come la telecamera Ulisse presente nel laboratorio, che può variare i suoi angoli di pan e di tilt. Si è quindi deciso di realizzare le funzioni `world2cam` e `cam2world` in modo che supportassero anche eventuali cambiamenti degli angoli di pan e di tilt della telecamera.

2.3.1 Prima implementazione

È importante notare che gli angoli di pan e di tilt della telecamera Ulisse rispetto alla posizione zero (quella in cui sia l'angolo di pan che l'angolo di tilt sono nulli) sono noti: ciò rende possibile l'utilizzo delle funzioni già descritte nei paragrafi 2.2.1 e 2.2.2 con la sola modifica della matrice di rototraslazione, in modo da tener conto delle due rotazioni aggiuntive.

La prima implementazione dell'algoritmo approssimava il cambiamento dell'angolo di pan con una rotazione attorno all'asse delle y della telecamera, e il cambiamento dell'angolo di tilt con una rotazione attorno all'asse delle x. Poiché, come descritto nei paragrafi precedenti, la rotazione di un angolo di tilt pari a ψ è rappresentata dalla matrice $R_x(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\psi) & \sin(\psi) \\ 0 & -\sin(\psi) & \cos(\psi) \end{bmatrix}$ mentre la rotazione di un angolo di pan pari a φ è rappresentata dalla matrice $R_y(\varphi) = \begin{bmatrix} \cos(\varphi) & 0 & -\sin(\varphi) \\ 0 & 1 & 0 \\ \sin(\varphi) & 0 & \cos(\varphi) \end{bmatrix}$, la rotazione complessiva si ottiene dal prodotto di tali matrici, ovvero $R_{xy} = R_x(\psi)R_y(\varphi)$.

La nuova matrice di rototraslazione veniva quindi semplicemente calcolata come il prodotto tra la matrice di rototraslazione derivata dalla calibrazione e la matrice di rotazione complessiva, ovvero:

$$A_{new} = R_{xy} \cdot A$$

Tuttavia il test di tale algoritmo ha dato risultati molto scoraggianti, in quando punti nell'immagine corrispondenti a determinate coordinate nel sistema mondo venivano mappati in punti di coordinate completamente diverse, introducendo errori per nulla trascurabili.

I risultati di tali test sono riassunti nella Tabella 2.5 e nella Tabella 2.6:

world2cam per Telecamere PTZ (Prima implementazione)										
		Coordinate globali (mm)			Pixel restituiti		Pixel reali		Errore (pixel)	
Pan	Tilt	x	y	z	x	y	x	y	x	y
-17	25	3000	2000	0	449	92	448	93	1	1
10	20	3000	2000	0	508	400	203	43	305	357
-5	35	3000	2000	0	338	229	339	214	1	15
-5	35	4000	2000	0	357	136	367	131	10	5
-20	40	3000	2000	0	283	59	473	254	190	195
-20	40	4000	2000	0	313	-58	486	165	173	?
-20	40	4000	2000	0	-22	-75	205	179	?	?

Tabella 2.5 - Risultati del test della prima implementazione dell'algoritmo world2cam sulla telecamera PTZ Ulisse.

cam2world per telecamere PTZ (Prima implementazione)											
		Pixel di input		Coordinate output (mm)			Coordinate reali (mm)			Errore (mm)	
Pan	Tilt	x	y	x	y	z	x	y	z	x	y
-17	25	449	92	3012	2013	0	3000	2000	0	12	13
-17	25	177	6	3904	3978	0	4000	4000	0	96	22
10	20	203	43	5860	3863	0	3000	2000	0	2860	1863
-5	35	339	214	3202	2030	0	3000	2000	0	202	30
-5	35	367	131	4091	1992	0	4000	2000	0	91	8
-20	40	473	254	-1367	-1834	0	3000	2000	0		
-20	40	486	165	1386	-639	0	4000	2000	0	2614	
-20	40	205	179	1053	2481	0	4000	4000	0	2947	2519

Tabella 2.6 - Risultati del test della prima implementazione dell'algoritmo cam2world sulla telecamera PTZ Ulisse.

La Figura 2.12 fa riferimento all'Ulisse avente angolo di pan pari a 10° e angolo di tilt pari a 20° ed esemplifica l'errore introdotto dalla funzione world2cam con in ingresso il punto di coordinate $(3000, 2000, 0)$:

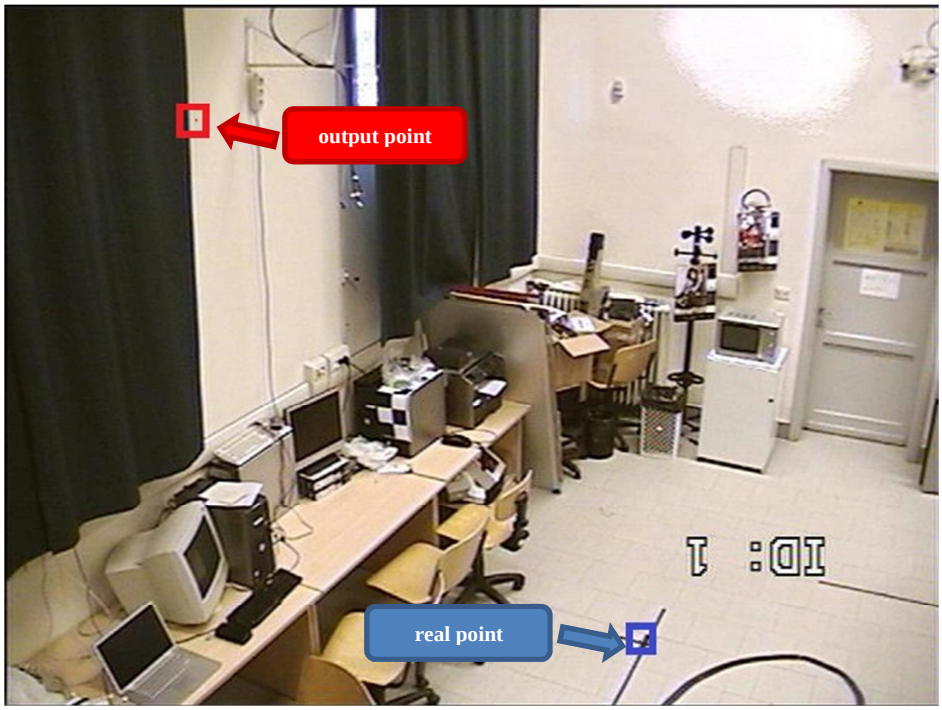


Figura 2.12 - Esempio di errore introdotto dalla prima implementazione della funzione `world2cam` per telecamere PTZ.

Dopo un'analisi del problema, si è visto che tali errori dipendevano dal fatto che, per quando riguarda la rotazione dell'angolo di pan, l'Ulisse non ruota attorno al suo centro ottico, bensì attorno ad un asse fisso rispetto al sistema di riferimento mondo. Per di più, poiché nella posizione con angoli di pan e di tilt pari a zero l'Ulisse non inquadrava il piano pavimento, la telecamera è stata calibrata in una posizione diversa (con angolo di pan pari a -17° e angolo di tilt pari a 25°), in modo da inquadrare il piano pavimento e quindi da poter vedere la scacchiera posta sullo stesso. Ciò comporta il fatto che nessuno degli assi della telecamera nella posizione di calibrazione sia parallelo all'asse di rotazione dell'Ulisse per quando riguarda l'angolo di pan; la rotazione per l'angolo di tilt è invece approssimabile con una rotazione attorno all'asse x della telecamera. Si è resa quindi necessaria l'implementazione di una funzione che tenesse conto di tali osservazioni.

2.3.2 Seconda implementazione

Si supponga quindi di essere nel sistema di riferimento della telecamera nella posizione di calibrazione. L'algoritmo implementato assume di conoscere un punto nell'asse di rotazione della telecamera: tale punto è stato misurato sperimentalmente ed è risultato avere coordinate $(7136, 3081, 0)$ nel sistema di riferimento mondo. Per ottenere le coordinate (X_U, Y_U, Z_U) del punto nel sistema di riferimento dell'Ulisse nella posizione di calibrazione è sufficiente moltiplicare il punto in coordinate del sistema di riferimento mondo per la matrice di

rototraslazione. La procedura utilizzata per ricalcolare la matrice di rototraslazione è quindi la seguente:

1. Si può notare che l'asse di rotazione dell'Ulisse è parallelo all'asse z del sistema di riferimento mondo: è quindi possibile ottenere il vettore di direzione $\vec{v}$ di tale asse nel sistema di riferimento della telecamera nella posizione di calibrazione applicando la matrice di rotazione al versore $[0 \quad 0 \quad 1]$.
2. Sia A il punto noto nell'asse di rotazione dell'Ulisse in coordinate del sistema di riferimento della telecamera. L'algoritmo applica una traslazione portando nel punto *A* l'origine del sistema di riferimento.
3. Successivamente, poiché l'asse di rotazione dell'Ulisse non è parallelo all'asse y del sistema di riferimento della telecamera, l'algoritmo applica un'isometria che porta quest'ultimo ad essere parallelo (e quindi a coincidere) con l'asse di rotazione della telecamera.

   Per calcolare la matrice di isometria l'algoritmo utilizzato è il seguente:

   - Sia $\vec{v} = [a \quad b \quad c]$ il vettore che rappresenta la direzione dell'asse di rotazione nel sistema di riferimento della telecamera. Per prima cosa occorre normalizzare tale vettore ottenendo il versore di direzione
     $$\vec{u} = \left[\frac{a}{\sqrt{a^2+b^2+c^2}} \quad \frac{b}{\sqrt{a^2+b^2+c^2}} \quad \frac{c}{\sqrt{a^2+b^2+c^2}}\right]$$
   - Per formare una base ortogonale, occorre trovare altri due versori ortogonali al versore $\vec{u}$. Sia quindi $\vec{u} = [x \quad y \quad z]$. È facilmente verificabile che un vettore ortogonale a quest'ultimo è dato dal vettore $[yz \quad xz \quad -2xy]$. Normalizzando tale vettore si ottiene il secondo versore $\vec{p}$. L'ultimo versore ortogonale si ottiene dal prodotto scalare dei primi due versori: $\vec{q} = \vec{p} \times \vec{u}$.
   - A questo punto si dispone di una base ortonormale $\{\vec{p}, \vec{u}, \vec{q}\}$. La matrice di isometria cercata è quindi data dalla matrice $S = \begin{bmatrix} p_1 & u_1 & q_1 \\ p_2 & u_2 & q_2 \\ p_3 & u_3 & q_3 \end{bmatrix}$.
4. A questo punto l'asse di rotazione dell'Ulisse è parallelo all'asse y del suo sistema di riferimento: è quindi possibile applicare la matrice di rotazione attorno all'asse y $R_y(\varphi) = \begin{bmatrix} \cos(\varphi) & 0 & -\sin(\varphi) \\ 0 & 1 & 0 \\ \sin(\varphi) & 0 & \cos(\varphi) \end{bmatrix}$, dove φ è l'angolo di pan desiderato.
5. L'algoritmo applica quindi la matrice di isometria inversa e successivamente il vettore di traslazione inverso a quello applicato al punto 2. Per quanto riguarda questo passo dell'algoritmo, si può notare come la matrice di isometria *S* sia una matrice ortogonale e pertanto la sua inversa coincide con la sua trasposta.

6. Infine l'algoritmo applica la matrice di rotazione attorno all'asse x dell'angolo di tilt, ovvero la matrice $R_x(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\psi) & \sin(\psi) \\ 0 & -\sin(\psi) & \cos(\psi) \end{bmatrix}$.

I passi appena esposti dell'algoritmo implementato sono riassunti in figura:

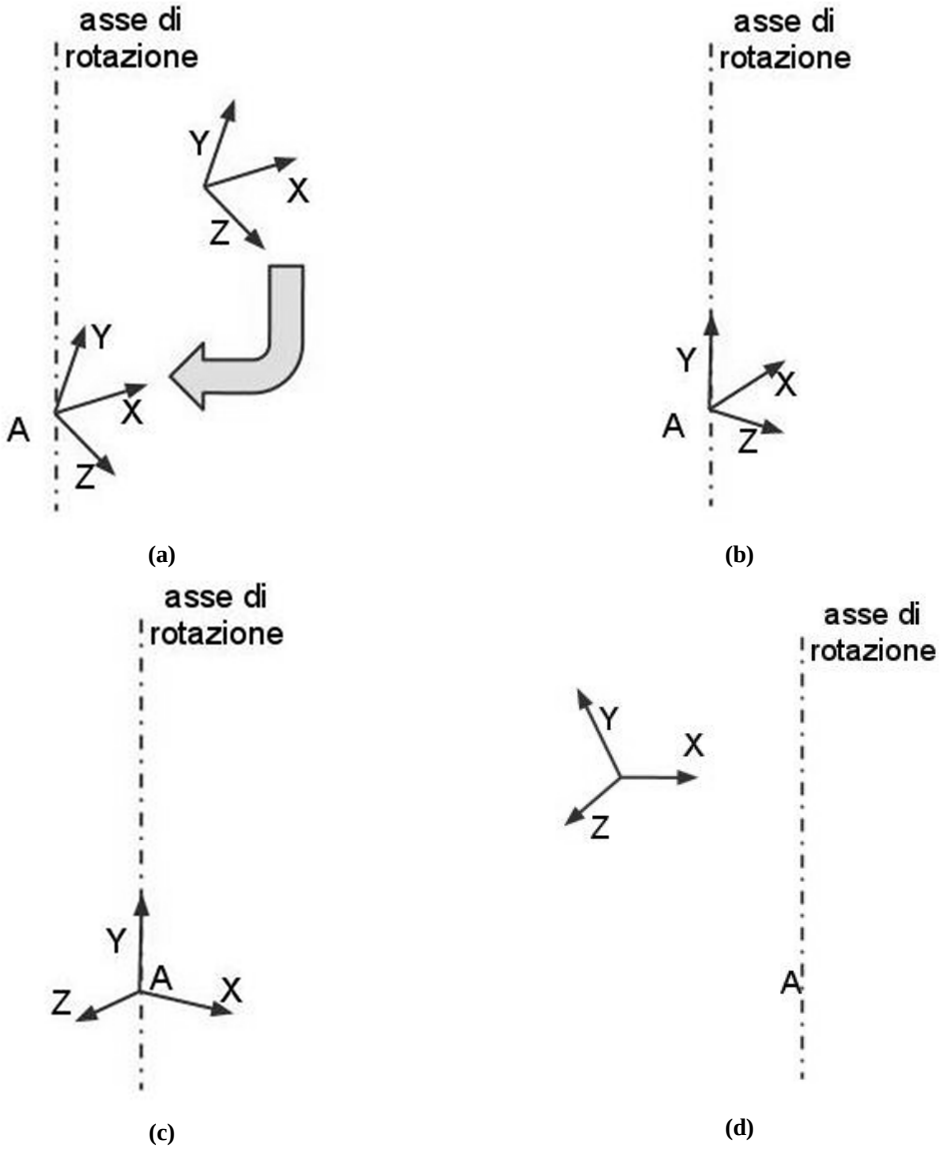


Figura 2.13 - (a) Traslazione del sistema di riferimento nel punto noto sull'asse di rotazione dell'Ulisse. (b) Applicazione della matrice di isometria. (c) Rotazione attorno all'asse y. (d) Applicazione della matrice di isometria inversa e della traslazione inversa.

Indicati con A la matrice di rototraslazione derivante dalla procedura di calibrazione, con T la matrice di traslazione nel punto noto sull'asse rotazione dell'Ulisse e con $(-T)$ la matrice di traslazione inversa, la matrice di rototraslazione complessiva dal sistema di riferimento mondo al nuovo sistema di riferimento della telecamera è quindi la seguente:

$$M = R_x(\psi) \cdot (-T) \cdot S \cdot R_y(\varphi) \cdot S^T \cdot T \cdot A$$

I risultati del test degli algoritmi appena descritti applicati alla telecamera PTZ Ulisse sono riassunti in Tabella 2.7 e in Tabella 2.8:

`world2cam` per Telecamere PTZ (Seconda implementazione)										
		Coordinate globali (mm)			**Pixel restituiti**		**Pixel reali**		**Errore (pixel)**	
Pan	**Tilt**	**x**	**y**	**z**	**x**	**y**	**x**	**y**	**x**	**y**
-17	25	3000	2000	0	449	92	448	93	1	1
10	20	3000	2000	0	196	38	203	43	7	5
-5	35	3000	2000	0	338	216	339	214	1	2
-5	35	4000	2000	0	370	128	367	131	3	3
-20	40	3000	2000	0	474	255	473	254	1	1
-20	40	4000	2000	0	493	161	486	165	7	4
-20	40	4000	2000	0	205	174	205	179	0	5

Tabella 2.7 - Risultati del test della seconda implementazione dell'algoritmo world2cam sulla telecamera PTZ Ulisse.

`cam2world` per telecamere PTZ (Seconda implementazione)											
		Pixel di input		**Coordinate output (mm)**			**Coordinate reali (mm)**			**Errore (mm)**	
Pan	**Tilt**	**x**	**y**	**x**	**y**	**z**	**x**	**y**	**z**	**x**	**y**
-17	25	449	92	3012	2013	0	3000	2000	0	12	13
-17	25	177	6	3904	3978	0	4000	4000	0	96	22
10	20	203	43	2981	1939	0	3000	2000	0	19	61
-5	35	339	214	3032	2000	0	3000	2000	0	32	0
-5	35	367	131	3980	2022	0	4000	2000	0	20	22
-20	40	473	254	3016	2019	0	3000	2000	0	16	19
-20	40	486	165	2971	2051	0	4000	2000	0	29	51
-20	40	205	179	3958	4006	0	4000	4000	0	42	6

Tabella 2.8 - Risultati del test della seconda implementazione dell'algoritmo `cam2world` sulla telecamera PTZ Ulisse.

La Figura 2.14 esemplifica l'errore introdotto dalla seconda implementazione della funzione `world2cam`. In particolare, la foto fa riferimento all'Ulisse nella stessa posizione di Figura 2.12 e alle stesse coordinate globali di input:

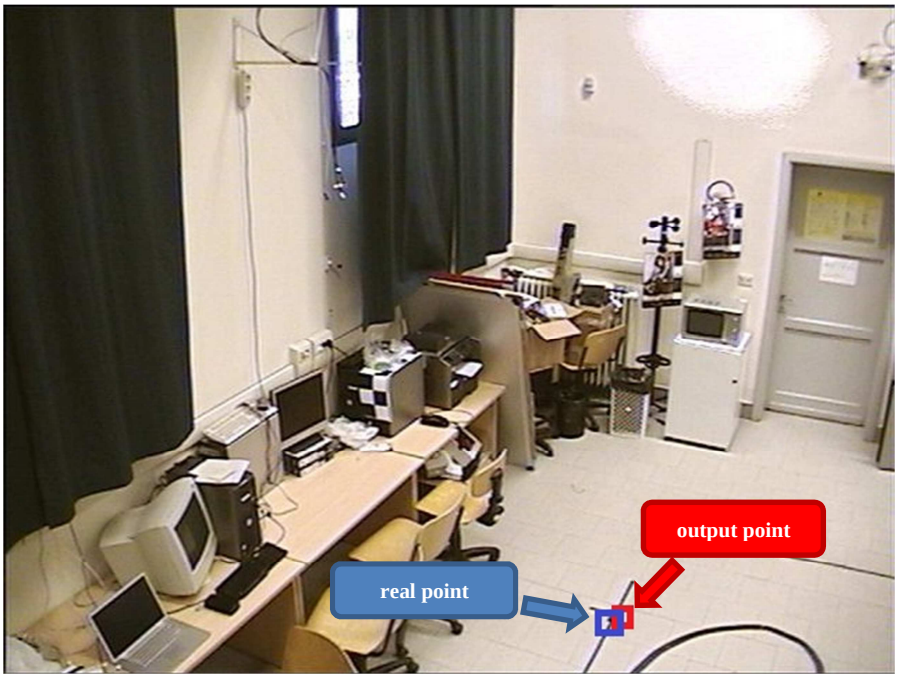


Figura 2.14 - Esempio di errore introdotto dalla seconda implementazione della funzione `world2cam` per telecamere PTZ

2.4 Calcolo degli angoli di pan e di tilt dell'Ulisse

Si è poi passati all'implementazione di un metodo che, ricevendo in ingresso le coordinate 3D di un punto nel sistema di riferimento mondo, fosse in grado di calcolare gli angoli di pan e di tilt da fornire alla telecamera Ulisse in modo tale che quest'ultima potesse centrare nel piano immagine tale punto.

2.4.1 Prima implementazione

La prima realizzazione della funzione assumeva di conoscere l'angolo di pan da fornire all'Ulisse in modo tale che l'asse z del suo sistema di riferimento risultasse essere parallelo all'asse x del sistema di riferimento del mondo. Tale angolo è stato misurato sperimentalmente ed è risultato essere pari a -17°.

L'algoritmo realizzato è quindi il seguente:

1. Dalla matrice di rototraslazione derivata dalla procedura di calibrazione, si calcola la posizione della telecamera nel sistema di riferimento del mondo. In particolare, tale posizione sarà rappresentata dal punto $U = (X_U, Y_U, Z_U)$.
2. Sia $P = (X_P, Y_P, Z_P)$ il punto fornito in ingresso in coordinate del sistema mondo. L'algoritmo assume che tale punto si trovi sul piano pavimento, ovvero che $Z_P = 0$. In questo caso, osservando la Figura 2.15, è possibile calcolare l'angolo di tilt attraverso le seguenti equazioni:

$$\overline{PU} = \sqrt{(X_U - X_P)^2 + (Y_U - Y_P)^2}$$

$$\alpha = \text{atan}\left(\frac{Z_U}{\overline{PU}}\right)$$

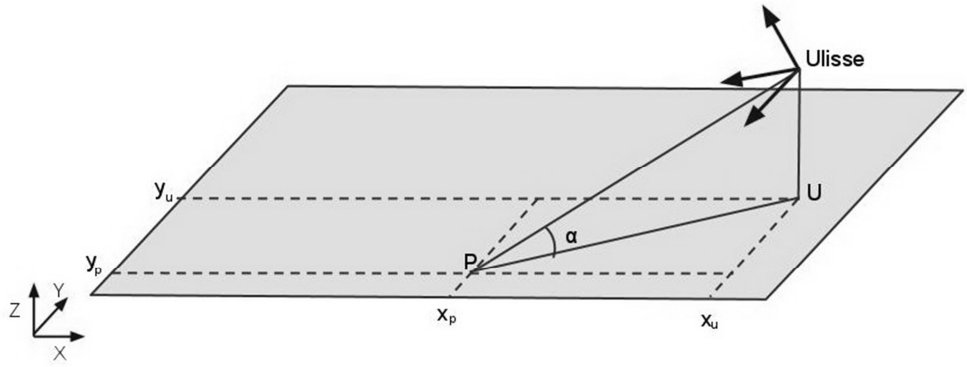


Figura 2.15 - Calcolo dell'angolo di tilt dell'Ulisse nella prima implementazione dell'algoritmo

3. Supponendo inizialmente che nella posizione con angolo di pan pari a zero l'asse z dell'Ulisse sia parallelo all'asse x del sistema di riferimento del mondo, e osservando la figura, l'angolo di pan è dato dalla seguente equazione:

$$\beta = atan\left(\frac{Y_U - Y_P}{X_U - Y_P}\right)$$

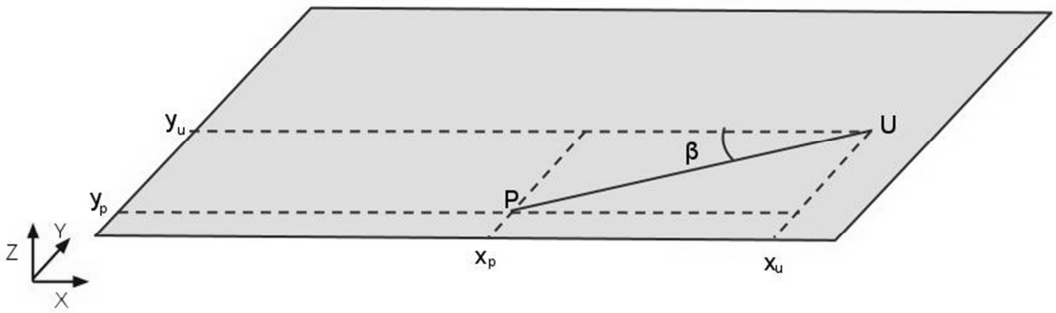


Figura 2.16 - Calcolo dell'angolo di pan dell'Ulisse nella prima implementazione dell'algoritmo.

Poiché tuttavia l'assunzione iniziale non è verificata, a tale angolo va aggiunto l'angolo misurato sperimentalmente, pari a -17°.

Testando l'algoritmo appena descritto, si è visto che tramite gli angoli di pan e di tilt forniti la telecamera Ulisse riesce sempre a visualizzare il punto desiderato nel proprio piano immagine, tuttavia tale punto non era quasi mai al centro dell'immagine. Ciò è dovuto a due assunzioni errate fatte dall'algoritmo appena esposto:

- la rotazione avviene attorno al centro ottico della telecamera;
- le rotazioni dovute agli angoli di pan e di tilt sono due rotazioni indipendenti.

Si è quindi scelto di cercare un'altra implementazione, che tenesse conto anche di queste osservazioni e che allo stesso tempo non avesse bisogno di conoscere in anticipo angoli misurati sperimentalmente.

2.4.2 Seconda implementazione

L'algoritmo realizzato come seconda implementazione della funzione è il seguente:

1. Attraverso l'algoritmo che ricalcola la matrice di rototraslazione dati gli angoli di pan e di tilt, esposta nel paragrafo 2.3.2, si calcola la matrice R di rototraslazione dell'Ulisse in posizione zero (angoli di pan e di tilt entrambi pari a 0).
2. Data quest'ultima matrice, e supponendo di conoscere un punto sull'asse di rotazione dell'Ulisse in coordinate del sistema di riferimento del mondo, l'algoritmo calcola le coordinate (X_A, Y_A, Z_A) di quest'ultimo punto nel sistema di riferimento della telecamera in posizione 0.
3. L'algoritmo calcola quindi la matrice di traslazione necessaria per portare l'origine del sistema di riferimento nel punto noto dell'asse di rotazione. Tale matrice è la seguente:

$$T = \begin{bmatrix} 1 & 0 & 0 & -X_A \\ 0 & 1 & 0 & -Y_A \\ 0 & 0 & 1 & -Z_A \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

   A questo punto è possibile calcolare la matrice complessiva A che consente di passare dal sistema di riferimento mondo al sistema di riferimento dell'Ulisse in posizione zero centrato sul punto noto dell'asse di rotazione:

$$A = T \cdot R$$

4. L'algoritmo calcola quindi le coordinate del punto di input in questo nuovo sistema di riferimento, rototraslando il punto in ingresso tramite la matrice A. Siano (X_I, Y_I, Z_I) le nuove coordinate ottenute.
5. A questo punto, come si può osservare dalla Figura 2.17, è possibile calcolare l'angolo di pan tramite la seguente equazione:

$$\psi = atan\left(\frac{X_I}{Z_I}\right)$$

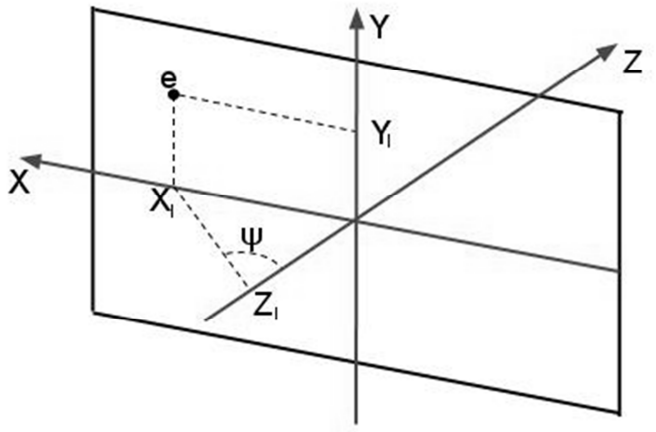


Figura 2.17 - Calcolo dell'angolo di pan dell'Ulisse nella seconda implementazione.

Una volta noto l'angolo di pan ψ, l'algoritmo calcola la matrice di rotazione dell'angolo di pan ψ attorno all'asse Y, ovvero $R_y(\psi) = \begin{bmatrix} \cos(\psi) & 0 & -\sin(\psi) \\ 0 & 1 & 0 \\ \sin(\psi) & 0 & \cos(\psi) \end{bmatrix}$, e la matrice che consente di effettuare un'operazione di traslazione opposta a quella effettuata nel punto 3, ovvero $T' = \begin{bmatrix} 1 & 0 & 0 & X_A \\ 0 & 1 & 0 & Y_A \\ 0 & 0 & 1 & Z_A \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

L'algoritmo calcola quindi una nuova matrice di rototraslazione, data da $M = T' \cdot R_y(\psi) \cdot T \cdot R$. Tale matrice consente di passare dal sistema di riferimento mondo al sistema di riferimento dell'Ulisse avente tilt pari a 0 e pan pari a ψ.

6. L'algoritmo effettua una rototraslazione del punto in input per la nuova matrice di rototraslazione, ottenendo così le coordinate del punto desiderato nel nuovo sistema di riferimento.

   A questo punto, come si nota in Figura 2.18, è possibile calcolare l'angolo di tilt tramite la seguente equazione:

$$\varphi = -atan\left(\frac{Y_I}{\sqrt{Z_I^2 + X_I^2}}\right)$$

   dove il segno negativo è dovuto al fatto che l'Ulisse utilizza per l'angolo di tilt un sistema di riferimento opposto a quello rappresentato in Figura 2.18, dove gli angoli di tilt positivi sono quelli che vanno "verso il basso".

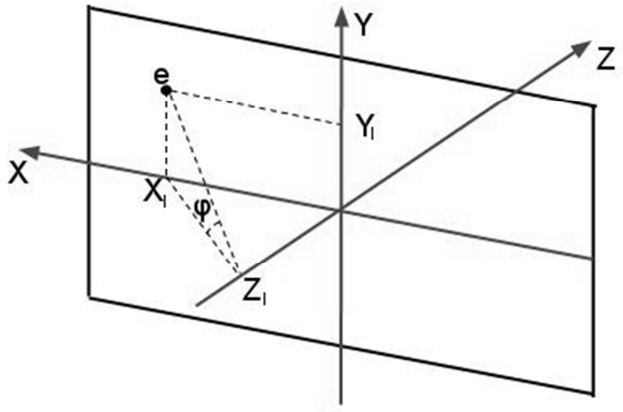
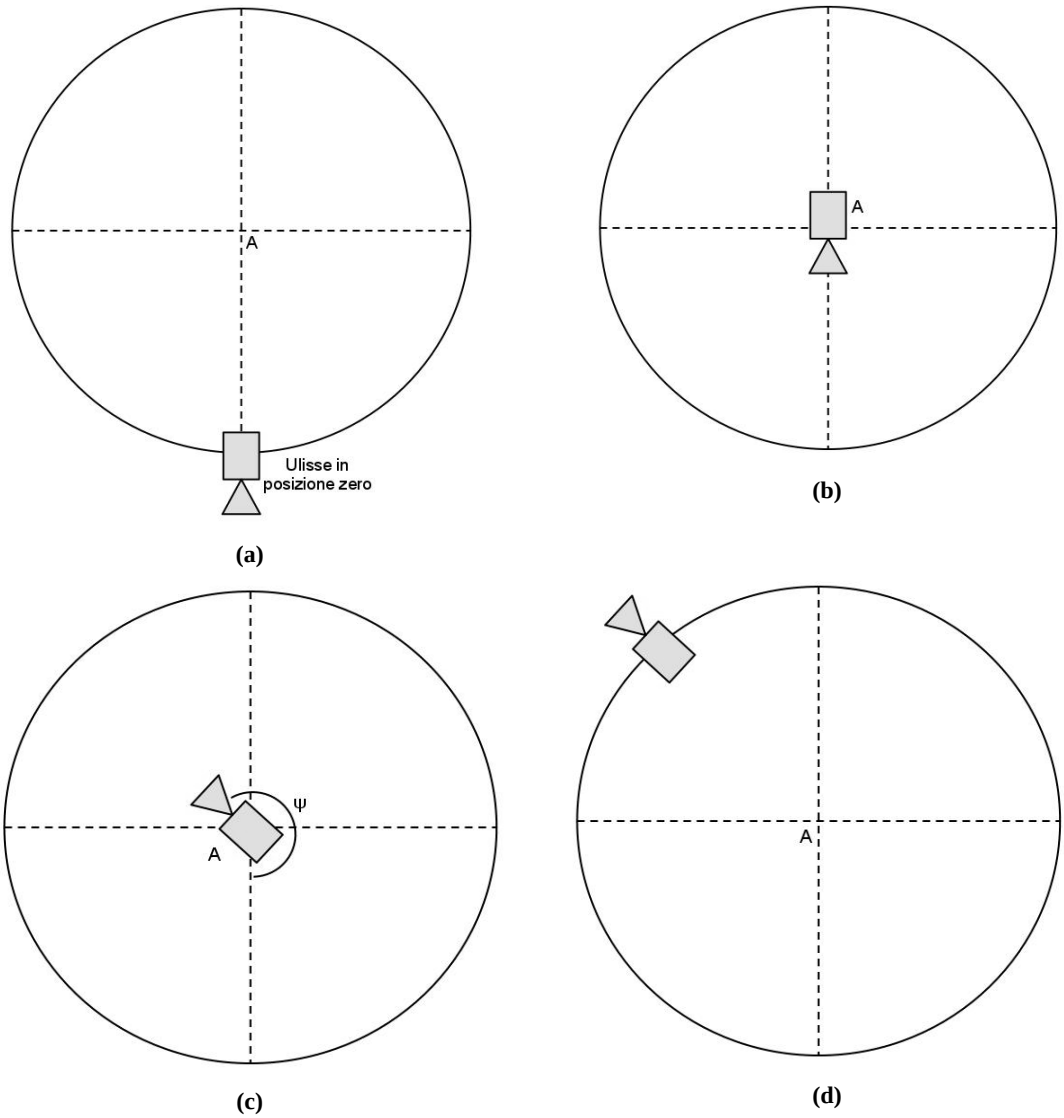
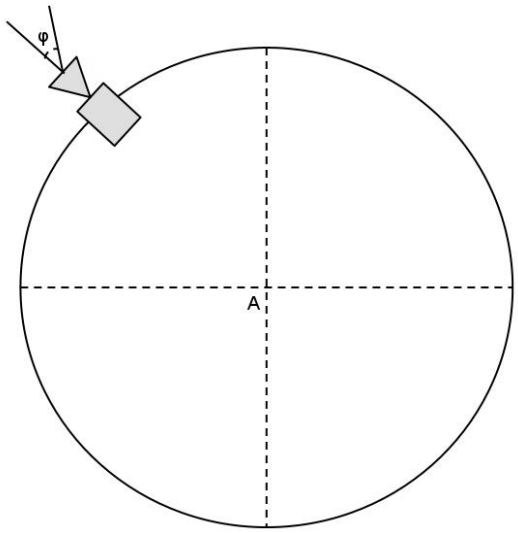


Figura 2.18 - Calcolo dell'angolo di tilt dell'Ulisse nella seconda implementazione.

La Figura 2.19 riassume l’algoritmo appena illustrato





(e)

Figura 2.19 - (a) Ulisse nella posizione zero. (b) Traslazione del sistema di riferimento nel punto noto sull'asse di rotazione dell'Ulisse. (c) Rotazione dell'angolo di pan. (d) Traslazione inversa del sistema di riferimento. (e) Rotazione dell'angolo di tilt

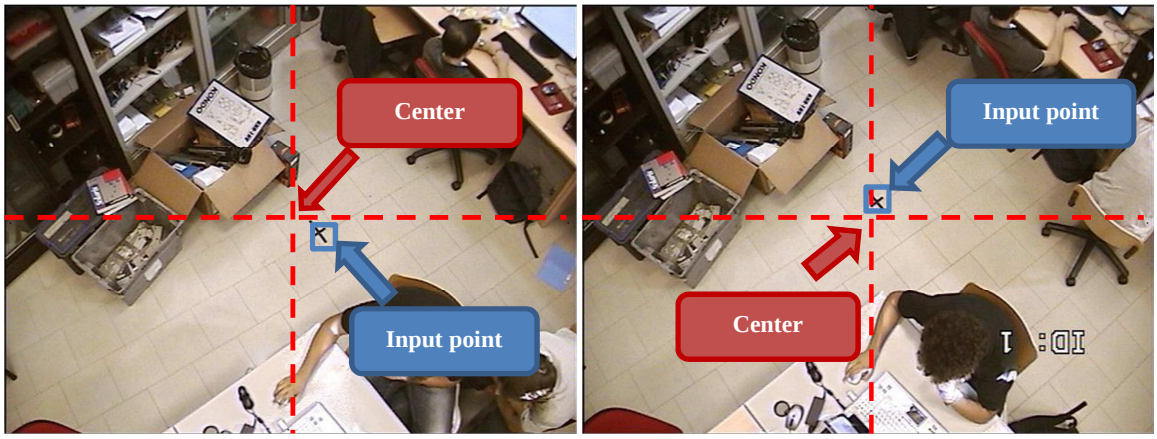
2.4.3 Confronto tra le due implementazioni

La Tabella 2.9 riassume i risultati del confronto tra le due implementazioni dell'algoritmo. In particolare si può notare come nella seconda implementazione dell'algoritmo il punto desiderato sia effettivamente più vicino al centro dell'immagine, corrispondente al punto $(322, 235)$, rispetto alla prima implementazione.

Confronto tra le implementazioni dell'algoritmo che calcola gli angoli di pan e tilt dell'Ulisse										
Coordinate punto (mm)			**Prima implementazione**		**Coordinate in pixel**		**Seconda implementazione**		**Coordinate in pixel**	
x	**y**	**z**	**pan**	**tilt**	**x**	**y**	**pan**	**tilt**	**x**	**y**
3000	2000	0	-1,93	38,52	311	260	-2,5	36,61	320	238
4000	2000	0	2,84	46,1	303	257	1,87	44,25	312	234
4000	4000	0	-34,7	46,46	323	252	-33,5	44,67	308	242
8620	2000	0	130,68	58,57	289	224	126,76	62,12	311	257

Tabella 2.9 - Confronto tra le due implementazioni dell'algoritmo che calcola gli angoli di pan e di tilt di una telecamera PTZ.

La Figura 2.20 evidenzia il discostamento del punto inserito come input dal centro dell'immagine nelle due implementazioni dell'algoritmo; nella figura si fa riferimento al punto di coordinate $(8620, 2000, 0)$, in quanto tale punto implica una variazione significativa degli angoli di pan e di tilt dell'Ulisse rispetto alla posizione di calibrazione. Il centro dell'immagine a cui si fa riferimento è quello ottenuto dalla calibrazione della telecamera e corrisponde al pixel di coordinate $(322, 235)$.



(a) **(b)**

Figura 2.20 - Esempio di discostamento del punto inserito in input dal centro dell'immagine nella prima (a) e nella seconda (b) implementazione dell'algoritmo.

2.5 Conclusioni

Questo capitolo ha descritto l'implementazione utilizzata per ottenere un modello 3D dell'ambiente utilizzabile da tutte le telecamere presenti nel sistema. Inoltre sono state implementate funzioni che considerano anche la presenza di telecamere PTZ nel sistema e che sono in grado di fornire parametri accurati per la gestione delle rotazioni delle stesse.

3 NMM (Network - Integrated Multimedia Middleware)

Questo capitolo ha lo scopo di fornire le nozioni di base riguardanti NMM, il middleware multimediale utilizzato in questo lavoro di tesi per gestire la distribuzione dei contenuti multimediali all'interno della rete locale.
La seguente introduzione ad NMM è in parte ispirata dalla documentazione reperibile dal sito ufficiale del software (17).

3.1 Introduzione ad NMM

Oltre ai PC, un numero sempre più crescente di dispositivi multimediali, come PDA e telefoni cellulari, dispone al suo interno di moduli per la connessione a reti di calcolatori. Tuttavia, la maggior parte delle infrastrutture multimediali moderne adottano un approccio centralizzato, in cui tutte le elaborazioni avvengono all'interno di un singolo sistema. La rete in questo caso viene utilizzata per lo più per inviare stream di dati multimediali dai client verso un server. Concettualmente, tali approcci consistono in due applicazioni indipendenti, server e client. La realizzazione di infrastrutture complesse diventa in questo caso difficile e fonte di errori, soprattutto dovuti soprattutto al limitato controllo che il client può esercitare sul server.

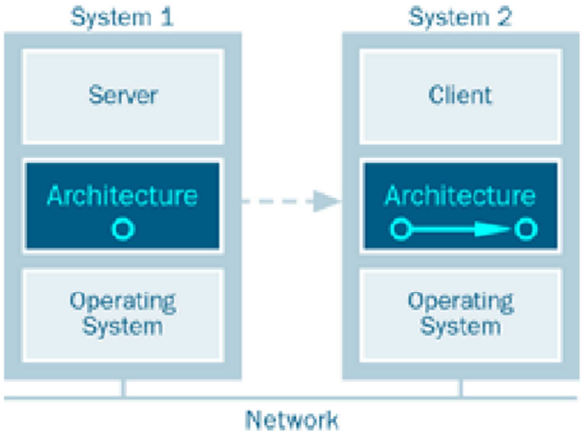


Figura 3.1 - Esempio di architettura client - server

NMM (Network-Integrated Multimedia Middleware) è pensato per superare tali limitazioni, fornendo un accesso a tutte le risorse interne alla rete: i dispositivi multimediali e i componenti software possono essere controllati in modo trasparente e integrati all'interno delle applicazioni. Contrariamente alla maggior parte delle architetture multimediali disponibili, NMM è un middleware, più specificatamente uno strato di software distribuito che viene eseguito tra i sistemi operativi e l'applicazione, come si può osservare anche dalla figura seguente:

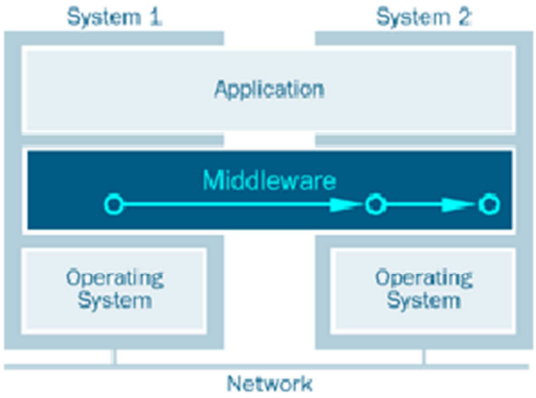


Figura 3.2 - Un middleware multimediale è uno strato di software distribuito che semplifica lo sviluppo di applicazioni

3.2 Nodi, jack e grafi di flusso

All'interno di NMM, tutti i dispositivi hardware e i componenti software sono rappresentati da entità chiamate *nodi*. Un nodo è caratterizzato da diverse proprietà che includono porte di input e output, dette *jack*, e formati multimediali supportati. Un formato definisce lo stream multimediale supportato dal nodo, ad esempio "audio/raw" per stream audio non compressi, con alcuni parametri aggiuntivi, come la frequenza di campionamento. Poiché un nodo può supportare più input o output, i suoi jack sono identificati da etichette.

A seconda della tipologia, un nodo può produrre dei dati, elaborarli o consumarli. I tipi di nodi attualmente presenti in NMM sono:

- Source node: produce dei dati e li invia agli altri nodi attraverso un jack di output.
- Sink node: consuma i dati che riceve dal jack di input.
- Converter node: composto da un jack di input ed uno di output, può modificare il formato dei dati o alcuni parametri di tale formato (ad esempio la risoluzione video).
- Filter node: può unicamente modificare i dati che giungono dal jack di input e fornirli in uscita tramite il jack di output, senza però cambiare formato o parametri dello stream.
- Multiplexer node: ha più jack di input ed un solo jack di output.
- Demultiplexer node: è dotato di un jack di input e più jack di output.

Queste tipologie di nodi possono essere connesse al fine di formare un grafo, in cui due jack connessi tra loro devono supportare lo stesso formato. La struttura di questo grafo specifica l'operazione che si vuole eseguire sui dati, come ad esempio la decodifica e la riproduzione di un file MP3.

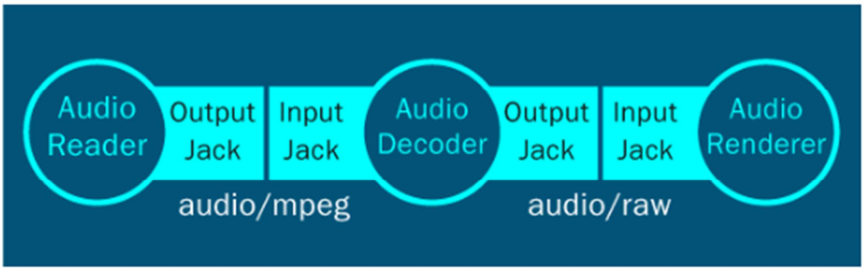


Figura 3.3 - Rappresentazione di un grafo per la decodifica e la riproduzione di un file MP3

Attualmente, NMM mette a disposizione più di 60 nodi già implementati, che consentono di integrare tra loro diversi dispositivi di input e output, codec o filtri specifici. Inoltre gli sviluppatori di NMM forniscono una documentazione volta a illustrare le modalità con cui è possibile creare dei propri nodi personalizzati.

3.3 Sistema di comunicazione

All'interno di NMM vengono definite due tipologie di messaggi:

- diretti: sono i dati multimediali che vengono copiati all'interno dei buffer;
- eventi: inoltrano informazioni di controllo, come ad esempio il cambiamento del volume degli altoparlanti. Gli eventi sono identificati da un nome e possono includere parametri di qualsiasi tipo.

Vi sono due diversi tipi di interazione all'interno di NMM, quella tra i vari nodi e quella tra NMM e l'applicazione. Per quanto riguarda la prima, i messaggi transitano attraverso jack connessi tra loro. Questo tipo di interazione è detta *instream* ed è per lo più utilizzata in una direzione che va dai nodi sorgente verso i nodi consumatori (*downstream direction*); tuttavia NMM consente anche di inviare messaggi nella direzione inversa (*upstream direction*).
Gli eventi *instream* sono di fondamentale importanza nei grafi multimediali: ad esempio la fine di uno stream può essere segnalata inserendo uno specifico evento al riempimento di un buffer. Inoltre possono essere registrati oggetti *listener* esterni per notificare gli eventi all'interno di un nodo (ad esempio per aggiornare la GUI alla fine di un file o per selezionare un nuovo file).
Gli eventi vengono anche utilizzati per un secondo tipo di interazione chiamato *out-of-band*, ovvero l'interazione tra l'applicazione e gli oggetti NMM, come i nodi o i jack. Gli eventi, in questo caso, sono utilizzati per controllare gli oggetti o per inviare notifiche dagli oggetti ai *listener* registrati.

3.4 Interfacce

NMM permette di definire le interfacce, descritte nell'NMM IDL (NMM Interface Definition Language), simile all'IDL di CORBA. Secondo tale stile di codifica, i nomi delle interfacce iniziano con la lettera maiuscola "I". Per ciascuna descrizione, un compilatore IDL crea

un’interfaccia e una classe per l’implementazione. Mentre, come si può capire anche dal nome, una classe per l’implementazione è utilizzata per implementare specifiche funzioni all’interno di un nodo, un’interfaccia può essere esportata al fine di interagire con altri oggetti.

3.5 Grafi di flusso distribuiti

Ciò che contraddistingue NMM è il fatto che i suoi grafi di flusso possono essere distribuiti all’interno della rete: dispositivi multimediali locali e remoti o componenti software incapsulati all’interno di nodi possono essere controllati ed integrati all’interno di un grafo multimediale comune, un grafo multimediale distribuito. Tale distribuzione è del tutto trasparente agli sviluppatori e non richiede alcun overhead aggiuntivo per le componenti locali del grafo.

La Figura 3.4 illustra un esempio di grafo distribuito per quando riguarda la riproduzione di file compressi, come i file MP3. Un nodo sorgente legge i dati dal file system locale, ed è connesso ad un altro nodo per la decodifica degli stream audio. Tale decoder è a sua volta connesso con un nodo che effettua il rendering del flusso audio non compresso verso una scheda audio. Una volta che il grafo viene eseguito, il nodo sorgente legge una certa quantità di dati dal file specificato, incapsula questi ultimi all’interno di buffer di dimensione prefissata e inoltra tali buffer al nodo successivo.

Come si può osservare dalla Figura 3.4, l’applicazione è eseguita all’interno del sistema System1, la lettura del file avviene all’interno del sistema System2 e la decodifica e la riproduzione avvengono all’interno del sistema System3.

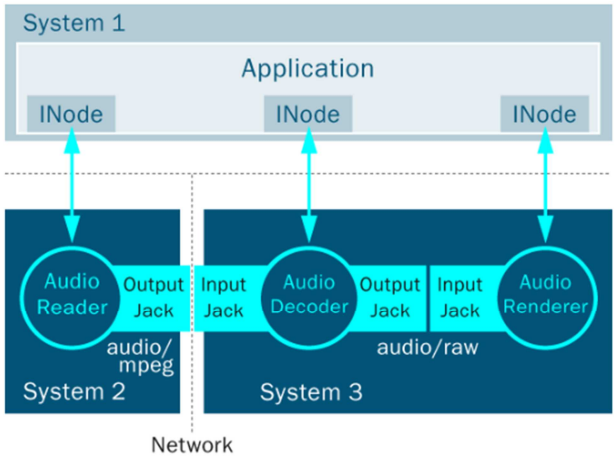


Figura 3.4 - Esempio di grafo distribuito per la riproduzione di file MP3.

NMM crea in modo automatico le connessioni di rete sia tra l’applicazione e i tre nodi distribuiti (*out-of-band interaction*), che tra il nodo sorgente e il nodo che effettua la decodifica (*instream interaction*). Successivamente, lo stream audio compresso secondo la codifica MPEG viene inviato all’interno della rete.

Si può facilmente notare come tale semplice grafo distribuito fornisce già degli importanti vantaggi. Innanzitutto, consente ad un'applicazione di accedere a file memorizzati all'interno del sistema distribuito senza la necessità di disporre di un file system distribuito. Inoltre, lo stream dei dati attraverso i nodi distribuiti connessi è gestito in maniera automatica da NMM; infine, l'applicazione agisce come un "controllore remoto" per tutti i nodi distribuiti. Ad esempio, questo consente di modificare in modo trasparente il volume di uscita della scheda audio remota semplicemente invocando un metodo di una specifica interfaccia, come `IAudioDevice`.

3.6 Sincronizzazione distribuita

Poiché i grafi di flusso di NMM possono essere distribuiti, essi consentono di effettuare il rendering di audio e video su diversi sistemi. Ad esempio, lo stream video di un file MPEG2 può essere visualizzato in un ampio schermo connesso ad un PC, mentre l'audio corrispondente può essere riprodotto all'interno di un dispositivo mobile.

Per realizzare la riproduzione sincronizzata dei nodi distribuiti all'interno della rete, NMM fornisce un'architettura di sincronizzazione distribuita. In aggiunta, gli stream multimediali possono essere riprodotti su più sistemi contemporaneamente. Un'applicazione comune è la riproduzione dello stesso stream audio utilizzando diversi dispositivi situati in diverse parti di un edificio.

La base per creare una sincronizzazione distribuita è l'utilizzo di una sorgente comune per ricavare informazioni sul tempo. NMM utilizza un clock statico che rappresenta il clock di sistema e che viene sincronizzato globalmente attraverso il protocollo NTP; esso pertanto può essere assunto rappresentare la stessa base temporale all'interno della rete.

3.7 Registry service

Il registry service di NMM consente la ricerca, la prenotazione e l'istanziazione dei nodi disponibili localmente e nei sistemi remoti. In ogni host un server registry amministra tutti i nodi disponibili all'interno dell'host stesso. Per ogni nodo, il server registry memorizza una sua descrizione completa che include la tipologia del nodo, il suo nome, le interfacce fornite e i formati di input e output supportati.

L'applicazione utilizza un registry client per inviare richieste ai registry server in esecuzione nel sistema locale o negli host remoti. I registry server sono contattati connettendosi a porte prestabilite. Dopo aver processato con successo la richiesta, il server registry riserva i nodi richiesti. I nodi sono poi creati o nell'host locale o nell'host remoto; per i nodi instanziati localmente, il registry client alloca gli oggetti all'interno dello spazio di indirizzamento dell'applicazione, al fine di evitare l'overhead dovuto alla comunicazione interprocesso.

Per creare un complesso grafo di flusso distribuito, l'applicazione può sia richiedere i nodi in maniera separata, sia utilizzare una descrizione del grafo. Tale descrizione include l'insieme delle descrizioni dei nodi connessi tramite i jack.
Affinché un'applicazione sia in grado di creare un grafo di flusso distribuito, l'applicazione di NMM chiamata serverregistry deve essere eseguita in ogni host che partecipa al flusso dei dati. Per applicazioni puramente locali tale requisito non è richiesto.

3.8 Clic – Un'applicazione per gestire i grafi multimediali in NMM

Il framework di NMM è utilizzato per progettare applicazioni multimediali. Come già detto nei paragrafi precedenti, i componenti di base in questo framework sono i nodi, ciascuno dei quali svolge una determinata funzione, come leggere un file video o decodificarlo. Tali nodi possono essere connessi tra loro in un grafo di flusso al fine eseguire una determinata attività, come ad esempio guardare un film o codificare un video in un altro formato.
L'obiettivo dei progettisti di NMM è stato quello di creare un'applicazione che, senza bisogno di interazioni con l'utente, richiedesse i nodi al server register, li connettesse formando un grafo di flusso e successivamente avviasse l'applicazione. L'applicazione *clic* ha esattamente questo compito: costruire un'applicazione NMM in modo automatico partendo da un file testuale che contiene la descrizione del grafo di flusso. Inoltre, tale applicazione consente anche di distribuire in modo automatico i nodi di NMM all'interno della rete locale.
La descrizione di un grafo di flusso è composta da due parti:

- la prima è un commento opzionale preceduto dal carattere %.
- la seconda specifica il grafo di flusso, ovvero come connettere tra loro i vari nodi.

L'esempio seguente descrive un grafo di flusso che riproduce un file .wav:

```
% This graph description realizes a simple WAV player.
% Use the -i option of clic to specify an WAV file
WavReadNode ! ALSAPlaybackNode
```

Per eseguire tale grafo, è sufficiente copiare il codice precedente all'interno di un file con estensione .gd (ad esempio wavplay.gd) ed eseguire il comando:

```
./clic wavplay.gd -i /home/bob/audio/song.wav
```

dove l'opzione `-i` serve a specificare il file di input.

3.8.1 Creare grafi di flusso

La creazione dei grafi avviene attraverso i nodi. Ogni nodo è identificato dal nome, che in generale corrisponde alla classe sorgente C++. Attraverso un punto esclamativo (!) si indica la connessione tra un nodo e l'altro. Pertanto attraverso l'istruzione

```
WavReadNode ! ALSAPlaybackNode
```

clic richiederà al server register i nodi `WavReadNode` e `AlsaPlayBackNode`. Se tutte le richieste vanno a buon fine, i due nodi vengono connessi tra loro. Infine clic imposta i parametri specificati attraverso la riga di comando e avvia l'applicazione.

Per specificare grafi di flusso più complessi che coinvolgano anche nodi di tipo demultiplexer e/o multiplexer, è necessario estendere questa sintassi in modo da poter descrivere e differenziare i vari jack di input e output che i nodi demultiplexer e multiplexer possono fornire. Per fare ciò, una stringa univoca chiamata *jack tag* è associata a ciascun jack al fine di identificarlo. Se un nodo fornisce un singolo jack di input o di output, è utilizzata la stringa "default" come jack tag.

È inoltre necessario estendere questa sintassi per descrivere rami di nodi che devono essere connessi ad uno specifico jack di output di un demultiplexer. Per specificare un ramo all'interno di un grafo di flusso, i nodi corrispondenti sono incapsulati tra parentesi graffe ({}), come si può osservare dall'esempio seguente, che descrive un grafo di flusso che legge file mpeg.

```
% This graph description realizes a simple mpeg video player with ac3
audio.
% Use the -i option of clic to specify the mpeg video file.
GenericReadNode
! MPEGDemuxNode
{
      { ["mpeg_video0"] ! MPEGVideoDecodeNode
      ! XDisplayNode }
      { ["ac3_audio0"] ! AC3DecodeNode
      ! ALSAPlaybackNode }
}
```

É inoltre prevista la possibilità di specificare il formato della connessione da utilizzare tra due nodi: per fare ciò si utilizza la parola chiave `Format` preceduta dal carattere `@`. Tale affermazione deve essere seguita da un elenco di tipi di formato separati da virgole e definiti all'interno del file `Format.hpp`. Un esempio dell'uso di tale affermazione è dato dal grafo seguente:

```
% This graph description converts an MPEG video file with AC3 audio
% into an AVI file. Furthermore the desired bitrate of the video is
% specified using the connection format.
% Use the -i option of clic to specify the mpeg video file and -o
option
% to specify the name of the output file.
GenericReadNode
! MPEGDemuxNode
{
      { ["mpeg_video0"] ! MPEGVideoDecodeNode
      ! FFMpegEncodeNode
      @ Format ("video/mpeg4", bitrate = 1200000)
      !
      ["video"]
      }
      {
      ["ac3_audio0"] !
      ["audio"]
      }
} AVMuxNode
! AVIWriteNode
```

Infine, è anche possibile specificare due parametri aggiuntivi che impostano il protocollo a livello di trasporto utilizzato per trasmettere i dati tra due nodi. Tale possibilità è particolarmente utile soprattutto se i nodi si trovano in host diversi. I due parametri che si possono impostare sono `setBufferStrategy` e `setEventStrategy`, entrambi preceduti dal carattere `@`. Un esempio dell'utilizzo di tali parametri è dato dal seguente grafo:

```
% This graph description describes a simple MP3 player which
specifies the
% used transport protocols between the nodes.
% Use the -i option of clic to specify the MP3 file
GenericReadNode @ setEventStrategy("TCPStrategy")
@ setBufferStrategy("RTPStrategy") !
MPEGAudioDecodeNode @ setBufferStrategy("RTPStrategy") !
PlaybackNode
```

3.8.2 Specificare i parametri dei nodi

In molti casi potrebbe essere necessario configurare alcuni parametri di un nodo, come la sua posizione all'interno della rete. La sintassi di clic consente di specificare tali parametri

preceduti dal simbolo '#'. Questi parametri devono essere specificati dopo il nome del nodo e i suoi *jack tag*.

Attualmente sono supportati i seguenti parametri:

- `setLocation(<string>)`: ha come argomento una stringa che specifica il nome dell'host o l'indirizzo IP della macchina in cui deve essere richiesto il nodo.
- `setPort(<int>)`: consente di specificare la porta da utilizzare per richiedere il nodo.
- `setSharingType(<sharing-type>)`: consente di specificare se il nodo può o non può essere riutilizzato da altre applicazioni.

L'unico parametro che verrà approfondito in quanto di interesse per questa tesi è `setLocation`.

L'esempio seguente illustra un grafo che richiede il nodo WavReadNode all'host avente indirizzo IP pari a 192.168.1.1:

```
% This graph description describes a simple WAV player where the
WavReadNode
% is running on host with IP address 192.168.1.1 . The serverregistry
on host with IP address % is listening on port 22801.
% Use the -i option of clic to specify the WAV file
WavReadNode # setLocation("192.168.1.1")
! ALSAPlaybackNode
```

3.8.3 Parametri di configurazione

In molti casi un nodo deve essere configurato prima di poter essere utilizzato: alcuni nodi richiedono, infatti, dei parametri aggiuntivi che non possono essere specificati tramite la riga di comando. In questo caso è necessario invocare un metodo specifico del nodo, definito nell'interfaccia NMM dello stesso. Ogni metodo presente in una interfaccia NMM può essere invocato dalla descrizione del grafo; per fare questo è necessario scrivere un corrispondente metodo IDL avente tutti gli argomenti preceduti dal carattere '$'.

La descrizione del grafo seguente mostra un esempio dell'utilizzo dei parametri di configurazione:

```
% This graph description describes a simple WAV player and shows how
to
% call interface methods from a graph description. It is not required
% to set the input file using the option -i of clic.
WavReadNode $ setFilename("/home/bob/audio/song.wav") INITIALIZED
! ALSAPlaybackNode $ setDownstreamMaxSize(1, "default") CONSTRUCTED
```

3.9 Installazione di NMM

Un rilevante periodo di tempo della mia tesi si è incentrato sull'installazione di NMM. Infatti, la procedura di installazione di questo middleware non è banale e la documentazione presente all'interno del sito ufficiale è incompleta e alcune volte anche errata.

Questo paragrafo ha quindi lo scopo di illustrare la procedura corretta per l'installazione di NMM, al fine di aiutare i prossimi utenti che dovranno utilizzare questo middleware multimediale.

La procedura di installazione può idealmente essere suddivisa in due passi:

- installazione dei prerequisiti
- installazione di NMM.

È importante notare come l'installazione dei prerequisiti richieda di disporre dei privilegi di amministratore sulla macchina; al contrario l'installazione di NMM non richiede tali privilegi e può essere eseguita da qualsiasi utente. Tuttavia, se, come in questa procedura, si installa NMM senza i privilegi di amministratore, esso sarà disponibile solamente per l'utente che ha effettuato l'installazione e non per tutti gli utenti del sistema.

La seguente installazione fa riferimento ad un sistema Ubuntu 10.10 a 64 bit.

3.9.1 Download delle librerie aggiuntive necessarie e dei sorgenti di NMM.

Per prima cosa, è necessario scaricare tutte le librerie aggiuntive necessarie per l'installazione di NMM. Nei repository di Ubuntu è presente gran parte di queste librerie. In particolare, è possibile installare tali librerie utilizzando il comando:

```
sudo apt-get install liba52-0.7.4-dev libfaad-dev ffmpeg libmp3lame-dev libraw1394-dev libmad0-dev libdvdnav-dev libdvdread-dev libogg-dev libvorbis-dev libshout3-dev fftw-dev liblivemedia-dev cdparanoia libpng12-dev libasound2-dev libx264-dev libjpeg62-dev imagemagick mplayer vlc transcode ogmtools libxml++2.6-dev expat openssl nasm libltdl-dev libavcodec-dev libavformat-dev libx11-dev libcdparanoia-dev libmpeg2-4-dev libssl-dev libxext-dev libxv-dev libexpat1-dev libmagick++-dev libfftw3-dev libssh-dev
```

Le librerie che devono essere compilate a mano e di cui quindi è necessario scaricare i sorgenti sono:

- libliveMedia: di questa libreria bisogna scaricare la versione presente nel sito di motama e non l'ultima versione, in quanto la prima contiene delle modifiche fatte da motama stessa. Il link è http://www.networkmultimedia.org//Download/external/nmm-2.2.0/external-libraries-optional/live.2009.06.02-14.tar.gz

- ulxmlrpcpp: di questa libreria va scaricata la versione 1.7.5 o successiva (la 1.7.4 non compila perché contiene un errore in un file sorgente). Il link è http://sourceforge.net/projects/ulxmlrpcpp/files/ulxmlrpcpp/

Ulteriori informazioni sulle librerie aggiuntive sono disponibili in http://www.networkmultimedia.org//Download/external/nmm-2.2.0/external-libraries-optional/index.htm.
Inoltre bisogna scaricare i sorgenti di NMM dal sito di motama.

3.9.2 Creazione dei link a faad

A questo punto è necessario creare in `/usr/local/include` la directory faad contenente i link `faad.h` e `neaacdec.h` che puntano ai relativi file in `/usr/include` . Questo è necessario perché NMM cerca quei file non nella directory di default `/usr/include`, bensì in `/usr/local/include/faad`, mentre in Ubuntu si trovano nella directory di default.

Per fare questo è necessario eseguire i seguenti comandi:

```
sudo mkdir -p /usr/local/include/faad

cd /usr/local/include/faad

sudo ln -s /usr/include/faad.h

sudo ln -s /usr/include/neaacdec.h
```

3.9.3 Installazione di libliveMedia

Per installare correttamente la libreria libliveMedia è necessario seguire la seguente procedura:

- Scompattare il pacchetto con il comando:

  ```
  tar xvf live.2009.06.02.tar.gz
  ```

- Entrare nella cartella appena creata, poi andare nella cartella `livemedia/include`, aprire il file `RTPSink.hh` e aggiungere la seguente riga dopo la riga 60 (se non è già presente):

  ```
  u_int32_t currentTimestamp() const { return fCurrentTimestamp; }
  ```

- Tornare nella directory principale della libreria, aprire il file `config.linux` e aggiungere a `COMPILE_OPTS` le opzioni `-DUSE_SYSTEM_RANDOM` e `-fPIC` (questa ultima va aggiunta solo se si sta usando un sistema a 64 bit).
- Generare i makefile con

  ```
  ./genMakefiles linux
  ```

- Compilare con `make`
- Compilare la libreria in modo dinamico eseguendo il comando:
  `gcc -fPIC -shared -o libliveMedia.so.2009.06.02 */*.a`

- In `/usr/local/include` creare la directory live eseguendo:

  ```
  sudo mkdir live
  ```

- Installare la libreria in `/usr/local/lib`: eseguire i seguenti comandi per copiare gli header all'interno di `/usr/local/lib`

  ```
  sudo find -name \*.hh -exec cp "{}" /usr/local/include/live ";"
  sudo find -name \*.h -exec cp "{}" /usr/local/include/live ";"
  ```

- Cambiare i permessi degli header della libreria appena creata (questa operazione è necessaria perché i permessi attuali consentono la lettura degli header della libreria solo all'utente root e se non vengono modificati l'installazione di NMM fallisce perché in questa guida viene installato in modo locale da un utente normale)

  ```
  sudo chmod -R 644 /usr/local/include/live/*.*
  ```

- Copiare la libreria compilata in /usr/local/lib e creare i link simbolici: eseguire i seguenti comandi:

  ```
  sudo cp libliveMedia.so.2009.06.02 /usr/local/lib
  cd /usr/local/lib
  sudo ln -s libliveMedia.so.2009.06.02 libliveMedia.so
  cd /usr/lib
  sudo ln -s /usr/local/lib/libliveMedia.so.2009.06.02
  libliveMedia.so
  ```

L'ultima istruzione è molto importante perché quando si compila NMM il programma va a cercare la libreria libliveMedia in `/usr/lib` e quindi è necessario creare un link a quella libreria in questa cartella. Inoltre in Ubuntu in `/usr/lib` è presente la libreria `libliveMedia.a` che è compilata in modo statico: senza questo link NMM cerca di utilizzare questa ultima libreria e la compilazione fallisce.

A questo punto la libreria libliveMedia è installata.

3.9.4 Installazione della libreria ulxmlrpcpp

Prima di installare la libreria ulxmlrpcpp è necessario installare delle librerie ausiliarie. Per fare questo è sufficiente eseguire i comandi

```
sudo apt-get install docbook-xsl fop doxygen libblitzOldbl libblitz-doc libblitz-dev
```

Si può quindi scompattare la libreria, entrare nella cartella e installarla tramite i comandi

```
./configure
```

```
make
```

```
sudo make install
```

3.9.5 Settaggio delle variabili d'ambiente

L'ultimo passo prima dell'installazione vera e propria di NMM è il settaggio di due variabili d'ambiente, la prima finalizzata ad impedire che il test della libreria ffmpeg fallisca e la seconda per impostare il percorso dove si trovano le librerie esterne che utilizzerà NMM. Per fare questo, è necessario eseguire i seguenti comandi:

```
export CPPFLAGS="-D__STDC_CONSTANT_MACROS"
```

```
export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
```

3.9.6 Installazione di NMM

A questo punto è finalmente possibile installare NMM. Per fare questo, occorre innanzitutto creare all'interno della nostra home la cartella `nmm` con il comando

```
mkdir ~/nmm
```

e successivamente copiare e scompattare al suo interno i sorgenti di nmm.

Si deve quindi eseguire la configurazione specificando il percorso dove installare NMM attraverso il comando:

```
./configure --with-extra-libs=/usr/local/lib --with-extra-
includes=/usr/local/include -–prefix=/home/carlo/nmm/nmm-2.2.0-
installed --enable-all
```

E infine si può installare il programma con

```
make
```

```
make install
```

Si noti che, poiché si sta installando NMM nella propria home, non è necessario eseguire il make install con i permessi di root.

3.9.7 Verifica dell'installazione

Se la procedura è stata eseguita correttamente, NMM è installato in ~/nmm/nmm-2.2.0-installed. Per verificare è sufficiente andare in ~/nmm/nmm-2.2.0-installed e digitare

```
./serverregisry -s
```

L'output dovrebbe essere il seguente:

```
serverregistry and Network-Integrated Multimedia Middleware (NMM) Version 2.2.0

Copyright (C) 2005-2010
Motama GmbH, Saarbruecken, Germany
http://www.motama.com

See licence for terms and conditions of usage

No plugin information available! If you start NMM for the first time this
information is created automatically.
Note: If you ever change your hardware configuration or update the NMM version you
must delete the file
/home/carlo/.nmm/plugins.2.2.0.casa._home_carlo_nmm_nmm-2.2.0-installed
or run 'serverregistry -s' again.

Create config file with plugin information ...
Loading plugins...
```

```
AACAudioDecodeNode  available
AC3DecodeNode  available
AC3ParseNode  available
ALSAPlaybackNode  available
ALSARecordNode  available
AVDemuxNode  available
AVMuxNode  available
AnalyseDataIdNode  available
BrightnessNode  available
BufferDropNode  available
BufferShapingNode  available
BufferTimestampControlNode  available
BufferTimestampNode  available
CDDANode  not available
CopyNode  available
DVBSimulatorNode  available
DVDNavReadNode  available
DVDReadNode  available
DevNullNode  available
DummyAudioSinkNode  available
DummyVideoSinkNode  available
FFMPEGDeinterlaceNode  available
FFMpegAVIReadNode  available
FFMpegAVIWriteNode  available
FFMpegAudioDecodeNode  available
FFMpegAudioEncodeNode  not available
FFMpegDecodeNode  available
FFMpegEncodeNode  available
FLACReadNode  available
FramerateConverterNode  available
GenericReadNode  available
GenericWriteNode  available
H264DecodeNode  available
IVTVReadNode  not available
IcecastNode  available
IdNode  available
JPEGDecodeNode  available
JPEGEncodeNode  available
LogoNode  available
M4AReadNode  available
MP3ReadNode  available
MPEGAudioDecodeNode  available
MPEGAudioEncodeNode  available
MPEGDemuxNode  available
MPEGReadNode  available
MPEGTSDemuxNode  available
MPEGTSReadNode  available
MPEGTimeShiftingNode  available
MPEGTimeShiftingNode2  available
MPEGVSHDetectionNode  available
MPEGVideoDecodeNode  available
MagickManipulationNode  available
MagickReadNode  available
MagickWriteNode  available
MessageSeekNode  not available
NetSinkNode  available
NetSourceNode  available
OGMDemuxNode  available
OSDManagerNode  available
OggVorbisDecodeNode  available
OverlayNode  available
PCMDecodeNode  available
PCMEncodeNode  available
PNGReadNode  available
PNGWriteNode  available
PlaybackNode  not available
RGBtoRGBConverterNode  available
RGBtoYV12ConverterNode  available
RawNode  available
RecordNode  not available
SAnalyzerNode  available
ScopeNode  available
TSDemuxNode  available
TVCardReadNode  not available
TVCardReadNode2  available
TimeDisplayNode  available
TimedBufferDropNode  available
URLNode  available
VISCACameraReadNode  not available
VideoCropNode  available
VideoGrabNode  available
VideoMuxNode  available
VideoScalerNode  available
WavReadNode  available
WavWriteNode  available
WhiteNoiseNode  available
XDisplayNode  available
YUVDeInterlaceNode  available
YUVtoRGBConverterNode  available
YUVtoYUVConverterNode  available
Finished loading plugins ...

Config file successfully written.
```

3.9.8 Nota per l'installazione per sistemi Ubuntu 11.04

Il codice sorgente di NMM presenta un errore di compilazione se si prova a compilarlo in una macchina avente come sistema operativo Ubuntu 11.04. Ciò è dovuto al fatto che NMM nel codice del nodo `tvCardReadNode` fa uso delle librerie `videoForLinux v1`, considerate obsolete e non più presenti nei repository di Ubuntu 11.04.

Tuttavia tale errore è facilmente risolvibile modificando il codice sorgente di NMM in modo da utilizzare le librerie `videoForLinux v2`; in particolare è necessario modificare il file `TvCardReadNode.hpp` cambiando la riga

```
#include <linux/videodev.h>
```

con la riga

```
#include <libv4l1-videodev.h>
```

3.9.9 Test dell'installazione

Al termine dell'installazione è possibile eseguire due test per verificare il corretto funzionamento del sistema.

Per prima cosa, si può eseguire tramite l'applicazione clic un semplice player audio. Per fare questo bisogna entrare nella directory di NMM e successivamente nella directory `apps/clic/gd/linux/playback/audio`, in cui si trovano dei file gd già pronti per poter essere eseguiti. È quindi sufficiente eseguirne uno, lanciando ad esempio il comando

```
clic wavplay.gd -i <file audio.wav>
```

Quest'operazione dovrebbe consentire di ascoltare il file specificato.

Il secondo test prevede la stessa procedura del primo tentando tuttavia di eseguire un video. Per fare questo è necessario entrare nella directory di NMM, successivamente nella directory `/apps/clic/gd/linux/playback/video` ed eseguire il comando

```
clic noise.gd
```

il quale dovrebbe visualizzare una finestra con del rumore video.

3.10 NMM SDK (NMM Software Development Kit)

Motama fornisce, in aggiunta ai sorgenti di NMM, un pacchetto software chiamato NMM SDK: esso non è altro che una semplice interfaccia finalizzata ad aiutare l'utente a sviluppare nodi personalizzati. I nodi relativi al progetto IDVS, e quindi anche quelli sviluppati in questa tesi, sono stati scritti inserendo il codice all'interno di questo pacchetto software. Il resto di questo paragrafo ha lo scopo di descrivere brevemente la procedura di installazione dell'sdk.

L'unico requisito necessario per l'installazione dell'sdk è ovviamente che nel sistema sia già presente un'installazione di NMM; nel seguito assumeremo di aver installato NMM all'interno della directory `/home/iaslab/nmm/nmm-2.2.0-installed/`.

Dopo aver scaricato ed estratto i sorgenti dal sito di motama, è necessario innanzitutto impostare la variabile di ambiente LD_LIBRARY_PATH in modo da includere non solo le

librerie esterne, ma anche le librerie installate con NMM. Per fare questo si utilizza il seguente comando:

```
export LD_LIBRARY_PATH=/home/iaslab/nmm/nmm-2.2.0-installed/lib/nmm:$LD_LIBRARY_PATH
```

Dopo essersi portati nella cartella principale dell'SDK, vanno eseguite le seguenti istruzioni, che hanno lo scopo di generare i makefile necessari per la compilazione:

- `./autogen.sh`
- `./configure --with-nmm=/home/bob/nmm-2.2.0-installed/ --prefix=/home/bob/nmm-2.2.0-installed/`

Infine, per completare l'installazione, è sufficiente compilare e in seguito installare i pacchetti tramite le istruzioni

```
make
make install
```

Si noti come, anche in questo caso, non bisogna inserire il comando sudo, in quanto l'installazione di NMM, e quindi anche dell'SDK, è stata effettuata solamente per l'utente corrente e non per l'intero sistema.
Per verificare che l'installazione sia andata a buon fine, è sufficiente eseguire il comando
`serverregistry –s.`
Se l'installazione è stata eseguita correttamente, l'esecuzione del serverregistry dovrebbe visualizzare i nostri nodi personalizzati in aggiunta ai nodi standard di NMM.[2]

3.11 Comunicazione tra nodi presenti in host diversi nella rete

Al termine dell'installazione di NMM si è passati a testare l'effettiva capacità di tale software di scambiare stream multimediali sulla rete, mettendo in comunicazione nodi presenti in host diversi.
Per utilizzare nodi che si trovano in un altro host, è innanzitutto necessario che in quest'ultimo sia attiva un'istanza del serverregistry. Per attivare un'istanza del serverregistry in un host è necessario eseguire il comando
`serverregistry`

[2] L'esecuzione del comando serverregistry –s non ha solamente lo scopo di verificare la correttezza dell'installazione, ma serve anche affinché NMM carichi correttamente tutti i nodi nel sistema. Poiché ogni volta che si effettua la modifica di un nodo personalizzato è necessario reinstallare l'SDK, dopo ogni installazione è necessario rieseguire anche questo comando, per consentire ad NMM di aggiornare le sue informazioni sui nodi installati.

Il cui scopo è porre un'istanza del programma in ascolto sulla porta 22801. Un comando equivalente è

```
serverregistry –d
```

Tuttavia quest'ultimo consente di visualizzare informazioni aggiuntive utili per eventuali debug.

Per utilizzare poi i nodi di un determinato host è sufficiente creare dei grafi di flusso distribuiti utilizzando il tag `setLocation`, come indicato nel paragrafo 3.8.2.

Tuttavia, l'esecuzione di tali grafi di flusso ha inizialmente presentato delle difficoltà; ciò era dovuto ad un problema di configurazione che riguardava i sistemi operativi Ubuntu 10.04 e Ubuntu 11.04. Per risolvere tale problema ed eseguire quindi l'applicazione distribuita, è necessario modificare con i permessi di amministratore il file di sistema `/etc/hosts` aggiungendo all'inizio una riga contente l'indirizzo IP e il nome del computer. Un esempio di file modificato è il seguente:

```
147.162.98.27 omni-pc
127.0.0.1   localhost.localdomain   localhost
::1   omni-pc     localhost6.localdomain6 localhost6
127.0.1.1   omni-pc
# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

dove in grassetto è stata evidenziata la nuova riga aggiunta. Ovviamente tale procedimento implica la necessità di modificare manualmente questo file ogni volta che l'host cambia il proprio indirizzo IP all'interno della rete.

3.12 Limiti di NMM

Questo paragrafo ha lo scopo di evidenziare i limiti e i difetti di NMM riscontrati durante questo lavoro di tesi, oltre a quelli già evidenziati nei paragrafi precedenti.

Innanzitutto, l'applicazione clic per la realizzazione dei grafi di flusso presenta dei limiti, in quanto non è in grado di creare grafi di flusso complessi, come quelli che ricevono input da varie sorgenti per poi indirizzarli all'interno di un unico nodo multiplexer al fine di elaborarli. In questo caso è necessario creare il grafo di flusso manualmente attraverso un programma scritto in codice C++.

NMM non è in grado di gestire la sincronizzazione dei vari input e output: già con un semplice nodo demultiplexer che riceve uno stream video in ingresso e ha due uscite diverse si è costretti ad eseguire il grafo in modalità non sincronizzata (in caso contrario l'esecuzione del grafo termina immediatamente con il lancio di un'eccezione). Inoltre, sempre a causa della mancanza di sincronizzazione, ci si può trovare ad elaborare dati di input diversi nei vari nodi del grafo. Ad esempio, si supponga di disporre di un nodo che legge un flusso video e di collegarci due nodi diversi, che effettuano elaborazioni differenti su quest'ultimo. La mancanza di sincronizzazione potrebbe far si che i due nodi elaborino in uno stesso istante frame differenti, propagando tale ritardo anche negli eventuali nodi successivi collegati.

NMM presenta poi molte incongruenze nella gestione degli eventi e nella documentazione degli stessi. Ad esempio, per l'invio di un evento in direzione upstream, NMM fornisce un metodo chiamato `sendEventUpStr`, di cui viene riportata la documentazione (reperibile nel sito ufficiale di Motama[3] (17)):

`Result sendEventUpStr(CEvent* cevent, const char* jack_tag = "default")`
`Send upstream composite event.`
`This method asynchronously sends a composite event in upstream direction from this node. The actual` **`CEvent`** `sent is a copy of the given` **`CEvent`**`.`
`If a null pointer is passed for the jack tag, the event is sent to all connected input jacks.`
`Parameters:`

`cevent`	`The composite event to send`
`jack_tag`	**`Jack`** `tag of jack over which the event is sent, or 0 to send to all jacks`

Tuttavia, se si specifica un jack di input il metodo ritorna `FAILURE`: infatti, analizzando il codice, si è scoperto che questa funzione per prima cosa verifica se effettivamente è presente un jack con il nome specificato. Tuttavia tale verifica non è effettuata tra i jack di input (come sarebbe ovvio, in quanto si vuole mandare un evento in direzione upstream), bensì tra i jack di output. L'unica soluzione è quindi chiamare un jack di input con lo stesso nome di un jack di output. Lo stesso errore è stato riscontrato per il metodo che invia gli eventi in direzione downstream. Questo bug è stato segnalato agli sviluppatori del software, tuttavia fino a questo momento non sono state fornite patch al riguardo.

Inoltre, anche specificando il nome del jack da utilizzare, NMM invia l'evento in tutti i jack presenti: pertanto se, come in questo lavoro di tesi, due nodi sono collegati tra loro da più di un jack, essi riceveranno più volte lo stesso evento. Per non eseguire quindi più di una volta la

[3] http://www.motama.com/doxygen-nmm-2.2.0/html/classNMM_1_1GenericNode.html#b0576c824d34d31c87c689538d7a69c7

stessa operazione, si è dovuto inserire all'interno di ogni evento inviato un numero intero generato in maniera casuale, al fine di identificare univocamente ciascun evento.
Un altro problema è stato riscontrato nell'invio di eventi composti: secondo la documentazione ufficiale, è possibile inviare più eventi inserendoli all'interno di una struttura contenitore chiamata `CEvent`. Tuttavia, l'inserimento di più di un evento all'interno di tale oggetto provoca la terminazione immediata del programma con l'errore `Segmentation Fault`; anche l'invio di più di una di tali strutture causa il medesimo errore. L'unica soluzione trovata a tale problema consiste nell'invio dei singoli eventi.
Infine, è stato riscontrato come la documentazione riguardante l'installazione del programma presenti degli errori e sia incompleta. Ad esempio, sebbene le librerie video4linux siano considerate deprecate, NMM fa ancora uso di tali librerie e non presenta documentazioni che illustrino come modificare il processo di installazione per quei sistemi operativi, come Ubuntu 11.04, che non le contengono più all'interno dei loro repository.
Il forum ufficiale del software è abbastanza frequentato dagli sviluppatori dello stesso e generalmente si ottengono risposte in tempi brevi; tuttavia queste ultime sono spesso molto brevi e hanno per lo più lo scopo di rimandare alla documentazione ufficiale del software. Nel caso di assistenza più approfondita, si viene in genere rimandati all'assistenza a pagamento.

4 Rilevazione di persone e di facce

L'algoritmo utilizzato in questo lavoro di tesi per effettuare la ricerca delle persone e delle facce all'interno di un'immagine è tratto dal lavoro di Viola – Jones; in particolare è stata utilizzata la sua implementazione presente all'interno delle librerie OpenCV.
Lo scopo di questo capitolo è fornire una breve introduzione all'algoritmo, rimandandone i dettagli alla lettura del paper corrispondente (18). Verrà inoltre presentata brevemente l'implementazione realizzata, ovvero verrà fornita una breve descrizione del nodo NMM realizzato per effettuare il riconoscimento delle persone e delle facce all'interno del flusso video proveniente da una telecamera.

4.1 Algoritmo di Viola – Jones

L'algoritmo di Viola – Jones è in grado di elaborare immagini in modo molto veloce, ottenendo al tempo stesso un'alta percentuale di successi. I principali contributi di questo algoritmo sono tre:

- Introduzione di un nuovo modo di rappresentare l'immagine, chiamato "Immagine Integrale", che consente di calcolare in modo estremamente rapido le feature utilizzate dal detector.
- Introduzione di un algoritmo di apprendimento basato su AdaBoost (19), che seleziona un piccolo numero di feature "critiche" ottenendo un classificatore molto efficiente.
- Introduzione di un metodo per comporre in cascata questi classificatori, consentendo di scartare quasi subito regioni dell'immagine appartenenti allo sfondo e di spendere invece più tempo di elaborazione nelle regioni più promettenti.

4.1.1 Estrazione delle feature

L'algoritmo di detection di Viola – Jones classifica le immagini basandosi sul valore di alcune semplici feature. I principali vantaggi derivanti dall'utilizzo delle feature piuttosto che dei singoli pixel sono il fatto che le prime consentono di codificare conoscenze specifiche del dominio di ricerca che sarebbero altrimenti difficili da apprendere attraverso un training set finito; inoltre un sistema basato sulle feature è molto più veloce di un sistema che opera sui singoli pixel.
L'algoritmo utilizza tre tipi di feature:

- *two – rectangle feature*: è la differenza tra la somma dei pixel all'interno di due regioni rettangolari.
- *three – rectangle feature*: calcola la somma dei pixel nei due rettangoli esterni, sottraendo da quest'ultima la somma dei pixel del rettangolo interno.

- *four – rectangle feature:* calcola la differenza tra coppie diagonali di rettangoli.

Le feature appena descritte sono rappresentate in Figura 4.1:

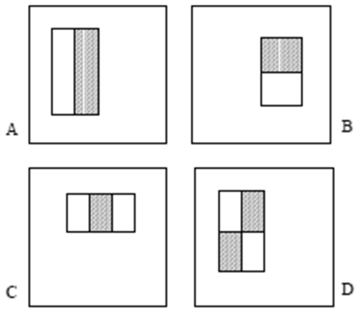


Figura 4.1 - Feature utilizzate dall'algoritmo Viola - Jones: A e B sono esempi di two-rectangle feature, C è un esempio di three-rectangle feature, D è un esempio di four-rectangle feature. (Immagine tratta dall'articolo ufficiale dell'algoritmo (18))

Le feature rettangolari possono essere calcolate in modo estremamente rapido utilizzando una rappresentazione alternativa dell'immagine chiamata "immagine integrale". Nello specifico, l'immagine integrale in posizione (x,y) contiene la somma dei pixel sopra e a sinistra del punto (x,y) nell'immagine originale, ovvero:

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

dove *ii(x,y)* è l'immagine integrale e *i(x,y)* è l'immagine originale. La Figura 4.2 esemplifica meglio la corrispondenza tra immagine integrale e immagine originale:

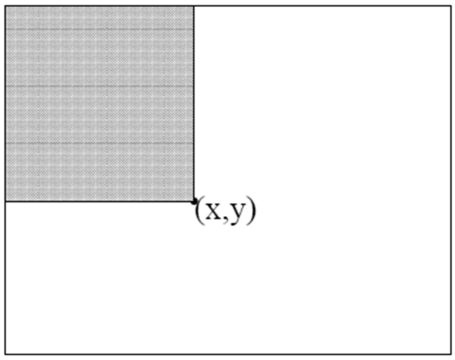


Figura 4.2 - Il valore dell'immagine integrale nel punto (x,y) è la somma di tutti i pixel sopra e sinistra del punto stesso. (Immagine tratta dall'articolo ufficiale dell'algoritmo (18)).

Come esemplificato in Figura 4.3, è importante notare che, utilizzando l'immagine integrale, ogni somma rettangolare può essere calcolata in quattro passi. Poiché le feature utilizzate coinvolgono rettangoli adiacenti, è possibile calcolare una *two – rectangle feature* in sei passi, una *three – rectangle feature* in otto passi e una *four – rectangle feature* in nove passi.

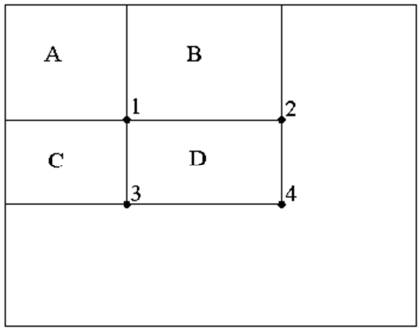


Figura 4.3 - La somma dei pixel all'interno del rettangolo D può essere calcolata in quattro passi. Infatti, il valore dell'immagine integrale nel punto 1 è la somma dei pixel del rettangolo A, il valore in posizione 2 è A+B, in posizione 3 è A+C e in posizione 4 è A+B+C+D. Pertanto D = 4 + 1 – (2 + 3). (Immagine tratta dall'articolo ufficiale dell'algoritmo (18)).

4.1.2 Funzioni di classificazione

Poiché vi sono 45.396 feature associate ad ogni sotto-finestra dell'immagine, sebbene il calcolo della singola feature sia molto rapido, calcolarne l'interno insieme non è computazionalmente fattibile. Lo scopo dell'algoritmo è quindi quello di trovare il minor numero di feature che, combinate insieme, formino un classificatore efficiente.

L'algoritmo utilizza una variante di AdaBoost sia per selezionare le feature che per allenare il classificatore. Nella sua forma originale, AdaBoost unisce tra loro una serie di funzioni di classificazione "deboli" ottenendo un classificatore finale sufficientemente robusto.

In pratica, ciascun singolo classificatore viene ristretto all'insieme delle funzioni di classificazioni che dipendono ciascuna da una singola feature. L'algoritmo "debole" di classificazione è quindi progettato per scegliere la singola feature rettangolare che meglio separa gli esempi positivi dagli esempi negativi. Per ogni feature, il classificatore debole determina la soglia ottimale della funzione di classificazione, per la quale il minor numero di esempi viene classificato in modo errato. Un classificatore debole ($h_j(x)$) consiste quindi in una feature (f_j), una soglia (θ_j) e un valore di parità (p_j) che determina il segno della disuguaglianza:

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

dove x è una sotto-finestra dell'immagine.

Nella pratica nessuna feature può eseguire l'attività di classificazione con un basso tasso di errore. Le feature selezionate per prime dall'algoritmo hanno tassi di errore compresi tra 0,1 e 0,3 mentre quelle selezionate al termine di quest'ultimo hanno tassi d'errore compresi tra 0,4 e 0,5.

Viene infine riportato l'algoritmo di apprendimento utilizzato da Viola – Jones:

- Siano date delle immagini di esempio $(x_1, y_1), \ldots, (x_n, y_n)$, dove $y_i = 0, 1$ indica, rispettivamente, se l'esempio è positivo o negativo.

- Si inizializzano i pesi $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ per $y_i = 0, 1$ rispettivamente, dove m e l sono rispettivamente numeri positivi e negativi.
- Per $t=1, \dots, T$:
    1. Si normalizzano i pesi
    $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
    in modo tale che w_t sia una distribuzione di probabilità.
    2. Per ogni feature j si allena un classificatore che sia limitato ad utilizzare un'unica feature. L'errore viene valutato rispetto a w_t, ovvero $\epsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$.
    3. Si sceglie il classificatore, h_t, con il minor numero errore ε_t.
    4. Si aggiornano i pesi:
    $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$
    dove $e_i=0$ se l'esempio x_i è classificato correttamente, $e_i=1$ in caso contrario; inoltre $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- Il classificatore robusto finale è
$$h(x) = \begin{cases} 1 & se \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & altrimenti \end{cases}$$
dove $\alpha_t = \log \frac{1}{\beta_t}$.

4.1.3 Cascata di classificatori

Si vuole infine descrivere brevemente l'algoritmo tramite il quale è possibile creare una cascata di classificatori in grado di ottenere ottime performance di rilevamento degli oggetti e nello stesso tempo di diminuire drasticamente il tempo di computazione necessario. L'idea chiave è quella di creare una cascata di classificatori in cui all'inizio siano presenti classificatori semplici che scartino il maggior numero di sotto-finestre, per utilizzare poi classificatori più robusti al fine di limitare al minimo il numero dei falsi positivi. In questo modo si genera un albero decisionale, che viene chiamato "*cascata*". Un risultato positivo dal primo classificatore richiede l'esecuzione del secondo classificatore, un risultato positivo dal secondo l'esecuzione del terzo e così via. Un risultato negativo in un qualsiasi punto della cascata provoca l'eliminazione immediata della sotto-finestra.

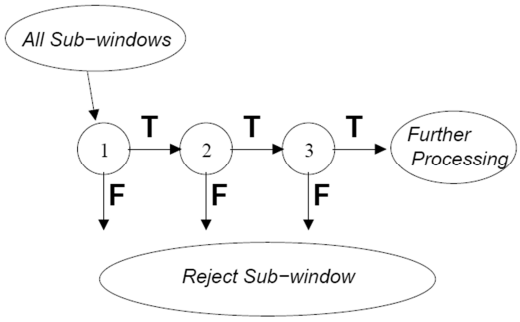


Figura 4.4 - Schema di una cascata per il rilevamento degli oggetti. I classificatori iniziali eliminano gran parte degli esempi negativi richiedendo poco tempo computazionale. (Immagine tratta dall'articolo ufficiale dell'algoritmo (18)).

La struttura della cascata riflette il fatto che, all'interno della singola immagine, la maggior parte delle sotto-finestre è negativa. Pertanto, lo scopo della cascata è quello di eliminare la maggior parte delle sotto-finestre negative nel più breve tempo possibile.

4.2 PeopleFaceDetectionNode

Al fine di rilevare le persone, e successivamente le loro facce, presenti in uno stream video proveniente da una telecamera è stato realizzato un nodo all'interno di NMM, chiamato `PeopleFaceDetectionNode`. Lo scopo di questo nodo è quello di rilevare l'eventuale presenza di persone all'interno della stanza, calcolarne le coordinate rispetto al sistema di riferimento mondo utilizzando il modello descritto nel Capitolo 2, rilevare la faccia della persona e inviare quest'ultima e le coordinate ad un nodo centrale, descritto nel Capitolo 6.
Il nodo realizzato è un demultiplexer / multiplexer, con due jack di ingresso e cinque jack di uscita. I jack di ingresso e uscita sono rappresentati in Figura 4.5:

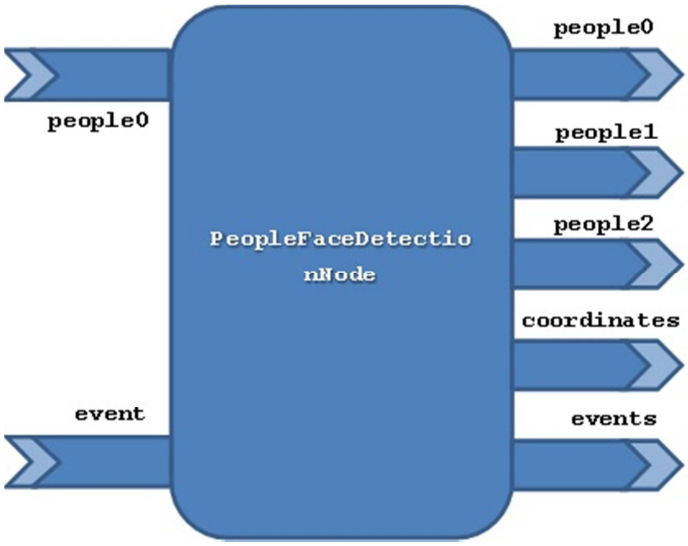


Figura 4.5 - Rappresentazione dei jack di ingresso e di uscita del nodo PeopleFaceDetectionNode. Il jack di ingresso *event* e il jack di uscita *events* sono attivi solamente se la telecamera che sta inviando il flusso video è una telecamera PTZ.

Il flusso video in ingresso viene ricevuto dal jack di input *people0*: il nome di questo jack è dovuto ad un bug di NMM, descritto nel Paragrafo 3.12, per il quale è necessario avere un jack di ingresso con lo stesso nome di un jack di uscita. Nei tre jack di uscita *people0*, *people1* e *people2* vengono inviate le facce delle persone presenti all'interno della stanza, mentre nel jack di uscita "*coordinates*" vengono inviate le coordinate di queste ultime espresse nel sistema di riferimento mondo globale. Nel caso il flusso video provenga da una telecamera PTZ sono presenti due jack aggiuntivi, il jack di ingresso *event* e il jack di uscita *events*, che hanno lo scopo di inviare e ricevere eventi riguardanti la gestione della telecamera stessa.

4.2.1 Elaborazione del flusso video in ingresso

Per il rilevamento delle persone e delle facce presenti all'interno di un'immagine è stata utilizzata l'implementazione dell'algoritmo di Viola – Jones presente all'interno delle librerie opencv. In particolare, il costruttore del nodo inizializza due oggetti di tipo `PeopleDetection` e `FaceDetection`, che hanno rispettivamente il compito di trovare le persone e le facce all'interno dell'immagine. Questi oggetti a loro volta caricano il modello corrispondente da un file xml reperibile all'interno delle librerie opencv e dispongono di un metodo `detect` che, data un'immagine, effettua il rilevamento tramite la funzione opencv `detectMultiScale` e restituisce un vettore di oggetti `cv::Rect` che rappresentano le bounding box delle persone o delle facce trovate.

Ogni volta che viene ricevuto un nuovo frame in ingresso, il nodo invoca il metodo `detect` dell'oggetto `PeopleDetection`, passando in ingresso il frame stesso. Al termine del rilevamento, per ciascuna persona individuata vengono svolte le seguenti operazioni:

- viene invocato il metodo `detect` dell'oggetto `FaceDetection`, passando in ingresso solamente la porzione dell'immagine che contiene la persona individuata;
- vengono calcolate le coordinate della persona nel sistema di riferimento mondo. Per calcolare tali coordinate, vengono utilizzate le funzioni `cam2world` descritte nei paragrafi 2.2.2 e 2.3.2 (a seconda che si stia utilizzando una telecamera fissa o una PTZ), fornendo in ingresso le coordinate in pixel del punto medio del lato inferiore della bounding box. La Figura 4.6 esemplifica meglio tale calcolo:

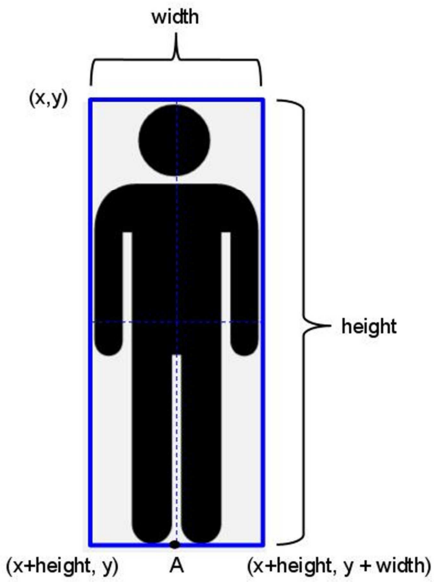


Figura 4.6 - Rappresentazione del calcolo del punto in pixel da fornire in ingresso alla funzione cam2world per ottenere le coordinate della persona trovata nel sistema di riferimento mondo.

Osservando sempre la Figura 4.6, le coordinate in pixel del punto A sono date dalla seguente equazione:

$$A \equiv \begin{cases} x_A = x + \dfrac{width}{2} \\ y_A = y + height \end{cases}$$

Infine, le eventuali facce trovate vengono inviate nei rispettivi jack di uscita. Per quanto riguarda le coordinate, esse vengono inviate nel jack "*coordinates*" all'interno di un vettore, ordinate secondo l'ordine di rilevamento. Al fine di indicare al nodo successivo se, per una persona rilevata è stata effettivamente rilevata, e quindi inviata, anche la sua faccia, il vettore contenente le coordinate ha come ultimo valore un flag. Tale flag è un numero espresso in codice binario, in cui l'i-esima cifra se pari a 0 indica che per la persona *i* non è stata individuata la faccia, se pari a 1 che per tale persona è stata individuata anche la faccia.

4.2.2 Gestione di una telecamera PTZ

Nel caso si stia utilizzando una telecamera PTZ, il nodo nel suo costruttore registra dei listener riguardanti tre eventi, "*ZOOM_ON_FACE*", "*ULISSE_IS_ZOOMING_P*", "*ULISSE_IS_NOT_ZOOMING_P*". Il primo evento indica che il nodo centrale ha deciso di zoomare sulla faccia di una persona presente all'interno della stanza, di cui vengono fornite le coordinate nel sistema di riferimento mondo. Nel caso di sua ricezione, il nodo `PeopleFaceDetectionNode` calcola gli angoli di pan e di tilt necessari utilizzando l'algoritmo descritto nel Paragrafo 2.4.2, lo zoom appropriato ed invia tali dati al nodo che controlla il movimento della telecamera PTZ.

La ricezione dell'evento *ULISSE_IS_ZOOMING_P* indica che la telecamera PTZ sta effettivamente zoomando sulla faccia della persona richiesta: in questo caso il nodo non è più in grado di effettuare il rilevamento delle persone e pertanto si disattiva. Allo stesso modo, la ricezione dell'evento *ULISSE_IS_NOT_ZOOMING_P* indica che la telecamera PTZ non sta più effettuando lo zoom su tale faccia, e provoca quindi la riattivazione del nodo `PeopleFaceDetectionNode`. Ulteriori dettagli in merito alla gestione degli eventi che regolano lo zoom della telecamera PTZ sono forniti nel paragrafo 6.2.1.

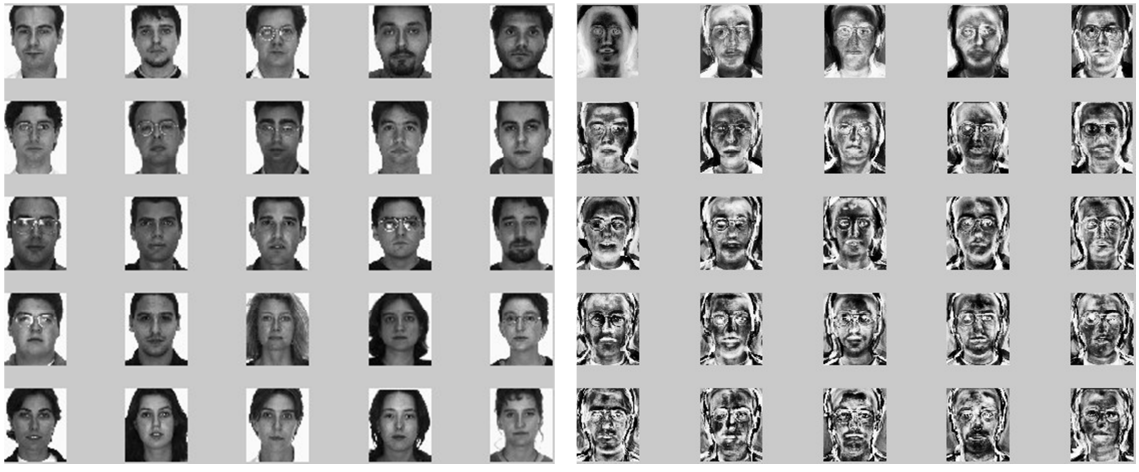
5 Introduzione al face recognition

L'obiettivo di questo capitolo è presentare l'algoritmo utilizzato per effettuare il face recognition. In particolare, verrà presentato l'algoritmo EigenFaces (5) utilizzato per l'estrazione delle feature, e successivamente verrà introdotto SVM (Support Vector Machine) (20), utilizzato per la classificazione di queste ultime. Infine, verrà presentata l'implementazione realizzata, ponendo l'accento anche sui limiti dell'algoritmo attualmente implementato e sui possibili miglioramenti futuri da effettuare.

5.1 Eigenfaces

L'algoritmo Eigenfaces, data l'immagine di una faccia da riconoscere, ne confronta le caratteristiche con quelle di facce di individui noti. In altre parole, l'algoritmo proietta le immagini delle facce in uno spazio delle feature che evidenzia le differenze significative tra le immagini delle facce. Le feature più significative sono chiamate eigenfaces, in quanto rappresentano gli autovettori dello spazio delle feature; esse inoltre generalmente non corrispondono a caratteristiche note della faccia, come occhi, orecchi, naso o capelli. L'operazione di proiezione caratterizza quindi ciascuna faccia come la somma pesata di questi autovettori, e pertanto per riconoscere una faccia è sufficiente confrontare tale somma con quella di individui noti.

Gli autovettori possono essere pensati come una serie di feature che insieme caratterizzano le differenze tra le varie immagini delle facce. Ciascuna parte dell'immagine contribuisce in maniera più o meno significativa a ogni autovettore, e pertanto è possibile visualizzare ciascuno di questi ultimi come una sorta di immagine "fantasma", chiamata nel seguito eigenface. In Figura 5.1 si può osservare un esempio di facce originali e relative eigenfaces:



(a) **(b)**

Figura 5.1 - Esempio di facce originali utilizzate in un training set (a) e relative eigenfaces (b).

La faccia di ciascuna persona può essere rappresentata esattamente come la combinazione lineare di eigenfaces; inoltre tale faccia può anche essere approssimata tramite la combinazione delle sole eigenfaces più significative. Ne segue che ciascuna persona può essere identificata da un insieme di pesi relativi alle feature, ovvero può essere descritta tramite una rappresentazione molto più compatta di un'immagine.

L'approccio appena descritto richiede che vengano effettuati i seguenti passi preliminari, al fine di preparare il modello con cui successivamente verrà eseguito il riconoscimento delle facce:

1. Acquisizione di un insieme iniziale di immagini di facce (detto training set).
2. Calcolo delle eigenfaces dal training set, mantenendo solo le M immagini che corrispondono agli autovettori più grandi. Le M immagini definiscono lo *spazio delle facce*.
3. Calcolo della distribuzione all'interno dello spazio M-dimensionale ottenuto al punto precedente dei pesi di ciascun individuo, proiettando l'immagine della sua faccia nello spazio delle facce.

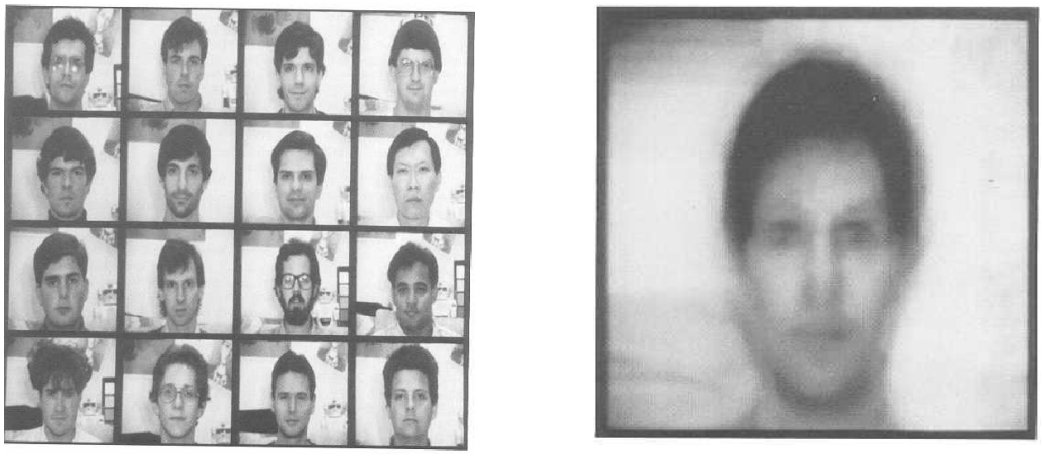
5.1.1 Calcolo delle eigenfaces

Si supponga che l'immagine di una faccia sia rappresentata da un array bidimensionale NxN di valori a 8 bit di profondità. Un'immagine può anche essere considerata come un vettore di dimensione N^2: ad esempio un'immagine di dimensione 256 per 256 può venire rappresentata da un vettore a 65536 dimensioni o, in modo equivalente, da un punto in uno spazio a 65536 dimensioni. Un insieme di immagini può quindi essere pensato come un insieme di punti in questo enorme spazio vettoriale. Le immagini delle facce tuttavia, avendo tutte una configurazione simile tra loro, non sono distribuite uniformemente in tale spazio e possono pertanto essere descritte da un sottospazio avente un numero inferiore di dimensioni. L'idea dell'analisi delle componenti principali è proprio quella di trovare quei vettori che rappresentano meglio la distribuzione delle immagini delle facce all'interno dell'intero spazio delle immagini. Tali vettori definiscono un sottospazio all'interno di quest'ultimo, che viene chiamato "spazio delle facce".

Si supponga di avere un training set di immagini di facce, indicate con Γ_1, Γ_2, Γ_3, … , Γ_M. La faccia media dell'insieme è definita come:

$$\psi = \frac{1}{M}\sum_{n=1}^{M} \Gamma_n$$

Ciascuna faccia differisce da quella media per il vettore $\Phi_i = \Gamma_i - \psi$. Nella Figura 5.2 si può osservare il training set utilizzato dagli autori dell'algoritmo e la relativa faccia media:



(a) **(b)**

Figura 5.2 - (a) Esempio di training set per il calcolo delle eigenfaces. (b) Faccia media ψ

Questo insieme di vettori di dimensioni molto elevate è quindi soggetto all'analisi delle componenti principali, che cerca un insieme di M vettori ortonormali, $\mathbf{u}_n$, che descriva nel modo migliore la distribuzione dei dati. Nello specifico, il k-esimo vettore u_k viene scelto in modo tale che

$$\lambda_k = \frac{1}{M}\sum_{n=1}^{M}\left(u_k^T \Phi_n\right)^2$$

sia un massimo, soggetto alla condizione

$$u_l^T u_K = \delta_{lk} = \begin{cases} 1 & se\ l = k \\ 0 & atrimenti \end{cases}$$

I vettori u_k e gli scalari λ_k sono rispettivamente gli autovettori e gli autovalori della matrice di covarianza

$$C = \frac{1}{M}\sum_{n=1}^{M} \Phi_n \Phi_n^T = AA^T$$

dove $A = [\Phi_1 \quad \Phi_2 \quad \cdots \quad \Phi_M]$. La matrice C tuttavia ha dimensione N^2, e il calcolo degli autovettori e degli autovalori diventa un'operazione intrattabile per immagini di dimensioni usuali. È quindi necessario un metodo computazionalmente fattibile per il calcolo di questi vettori. Tale metodo è possibile grazie all'analisi dei componenti principali, che riduce in maniera significativa la complessità dell'algoritmo: il calcolo degli autovettori diventa infatti O(M) anziché $O(N^2)$, con $M << N^2$.

5.1.1.1 Database Yale

Al fine di realizzare un algoritmo di riconoscimento in grado di operare in un dataset aperto, è possibile utilizzare dei database di facce standard per creare la base delle eigenfaces. In particolare in questa tesi è stato utilizzato lo Yale DB (21): il database contiene le facce di 15

persone, ciascuna in 11 pose diverse. Tutte le immagini sono della stessa dimensione e sono state elaborate al fine di toglierne lo sfondo. Un esempio del database è riportato in Figura 5.3:

Figura 5.3 - Una parte dello Yale DB

5.1.2 Utilizzo di Eigenfaces per classificare l'immagine di una faccia

Le immagini eigenfaces calcolate dagli autovettori formano una base con la quale è possibile descrivere immagini di facce. Nello specifico, l'attività del riconoscimento di facce viene trasformata in un'attività di riconoscimento di pattern. Le eigenfaces definiscono infatti un sottospazio di *M'* dimensioni rispetto allo spazio originale delle immagini a N^2 dimensioni.

Una nuova faccia viene trasformata nelle sue componenti eigenfaces (proiettata nello spazio delle facce) tramite l'operazione

$$\omega_k = u_k^T(\Gamma - \psi)$$

con k = 1, … , M'. I pesi formano un vettore $\Omega^T = [\omega_1 \quad \omega_2 \quad \cdots \quad \omega_{M'}]$ che descrive il contributo di ciascuna eigenface nel rappresentare l'immagine di ingresso. Tale vettore può quindi essere usato in qualsiasi algoritmo di riconoscimento di pattern per trovare il numero della classe, se esiste, che descrive meglio la faccia di ingresso. L'algoritmo più semplice in questo senso è quello che calcola la distanza euclidea tra il vettore dei pesi e quello di ciascuna immagine del training set, attribuendo quindi alla faccia di ingresso la classe per la quale è stata calcolata la distanza minima.

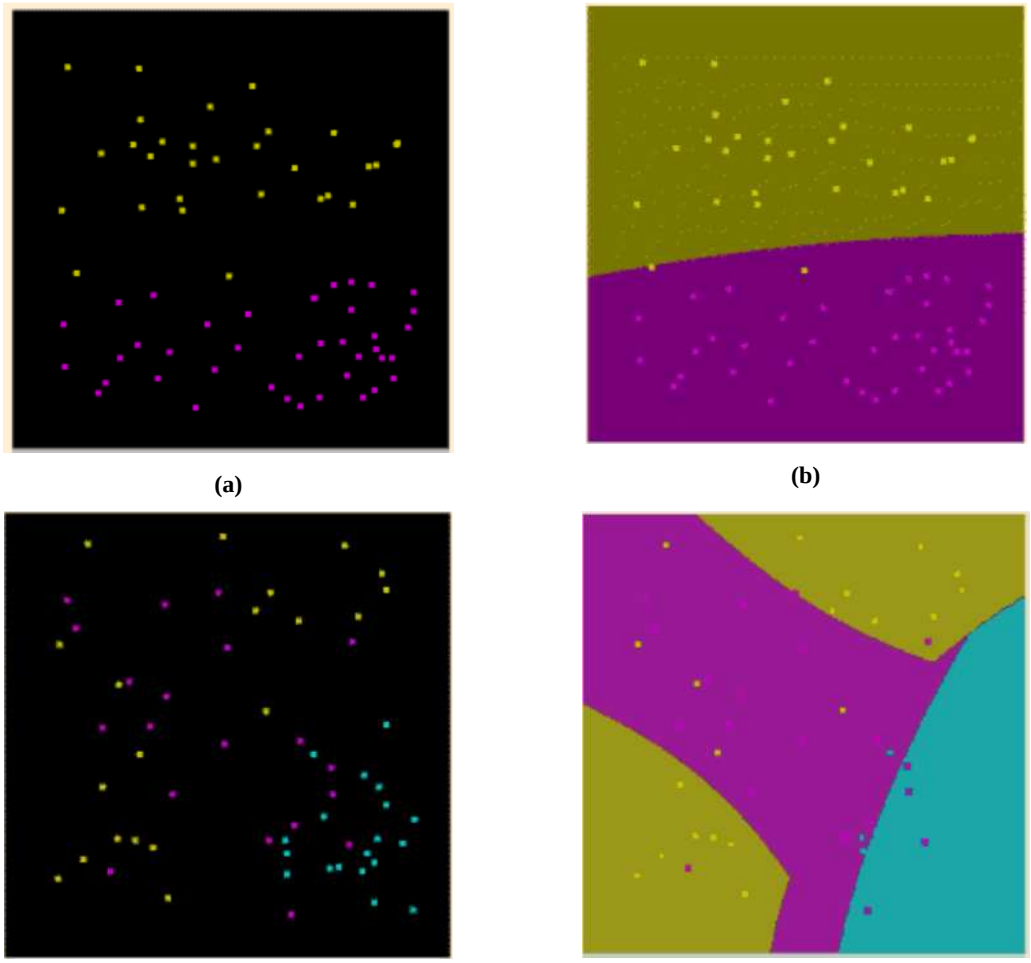
5.1.3 Limiti dell'algoritmo Eigenfaces

Nella descrizione precedente è stato ignorata la presenza dello sfondo. Nel concreto, la presenza dello sfondo può drasticamente ridurre le performance dell'algoritmo di riconoscimento, in quanto l'analisi delle eigenfaces appena descritta non distingue lo sfondo dalla faccia vera e propria. Per di più, lo sfondo potrebbe rappresentare una parte significativa dell'immagine usata per classificare le facce.

Questo problema è in parte risolvibile moltiplicando l'immagine di ingresso per una finestra gaussiana bidimensionale centrata sulla faccia, diminuendo in questo modo il contributo dello sfondo e accentuando quello della faccia stessa.
È stato inoltre dimostrato come le perfomance dell'algoritmo decrescano significativamente qualora si vari la dimensione della testa nell'immagine; quest'ultima deve infatti essere molto vicina alle dimensioni delle teste nelle eigenfaces affinché l'algoritmo ottenga dei buoni risultati. Una soluzione a tale problema è quella di scalare tutte le immagini portandole ad una dimensione pari a quella delle eigenfaces; in alternativa è possibile usare eigenfaces di più dimensioni e confrontare le immagini di ingresso con quella avente le dimensioni più vicine.
L'algoritmo è inoltre sensibile a variazioni nell'orientazione della testa. Tramite tecniche di visione computazionale è comunque possibile stimare l'orientazione della testa nelle immagini di ingresso e ruotare quest'ultima in modo da allinearla con quella delle eigenfaces.

5.2 SVM

In questo lavoro di tesi è stata usata SVM per creare un modello, a partire dalle feature estratte con l'algoritmo eigenfaces, per il riconoscimento di facce da utilizzare poi nella fase di riconoscimento.
Di seguito si fornisce una breve introduzione a tale tecnica: un'esposizione completa della teoria di SVM esula dagli scopi di questa tesi e si rimanda quindi alla relativa pubblicazione (20).
SVM è una tecnica utilizzata soprattutto recentemente per affrontare problemi di classificazione. In generale, dover risolvere un problema di classificazione significa disporre di un modello di dati precedentemente classificato e, tramite questo, dover predire a quale classe appartengono i nuovi dati ricevuti in ingresso. SVM è in grado di risolvere tale problema in tempi ragionevoli facendo uso della teoria dell'apprendimento automatico.
Tale tecnica può essere meglio esemplificata tramite un esempio. Si consideri una serie di punti di diversi colori in un piano: il colore di ciascun punto rappresenta la sua classe, mentre le posizioni dei punti sono i dati da analizzare. SVM per prima cosa cerca le equazioni per dividere tali punti, dividendo il piano in varie regioni colorate osservabili in Figura 5.4 (b) e (d):



(a) **(b)**

(c) **(d)**

Figura 5.4 - Esempio di utilizzo di SVM.

Tali regioni vengono poi usate per predire la classe (in questo caso il colore) dei nuovi punti in ingresso, utilizzando solo la loro posizione.

5.2.1 libsvm

L'implementazione di SVM utilizzata in questa tesi è `libsvm`, libreria open source scritta in linguaggio C++ e scaricabile gratuitamente al link http://www.csie.ntu.edu.tw/~cjlin/libsvm/ (22).

Una volta scaricato e compilato il tool, si ottengono tre eseguibili, che sono stati utilizzati per creare il modello usato successivamente nel riconoscimento di facce.

In particolare, gli eseguibili ottenuti sono:

- `svmtrain`: utilizzato per creare il modello tramite il training set. Questo programma accetta in ingresso un file contenente il training set codificato secondo uno specifico formato e genera un file contenente il modello.
- `svmpredict`: questo programma fa uso del modello creato tramite l'eseguibile precedente per predire la classe dei nuovi dati in ingresso.

- `svmscale`: viene utilizzato per scalare i dati. Questo è importante perché i dati iniziali potrebbero essere troppo grandi o troppo piccoli, ed è quindi necessario portarli all'interno di un intervallo che consenta di eseguire le fasi di training e di predict nel tempo più breve possibile.

Il formato secondo cui codificare i dati è il seguente:

```
[label] [index1]:[value1] [index2]:[value2] ...
[label] [index1]:[value1] [index2]:[value2] ...
```

…

5.2.1.1 Classificazione all'interno di libsvm

La classificazione implementata all'interno di libsvm è una classificazione multiclasse: in particolare è composta da una serie di classificazioni binarie pari al numero di possibili composizioni di coppie delle classi che compongono il modello. Ogni classificazione binaria fornisce come risultato una confidenza che rappresenta l'esito del confronto tra le due classi: in particolare essa è un numero positivo nel caso l'elemento è risultato essere più vicino alla prima classe, negativo in caso contrario.

La classe predetta dall'algoritmo di classificazione è quindi la classe che ha vinto più confronti; nel caso vi siano più classi che hanno vinto lo stesso numero di confronti viene restituita la prima di queste ultime.

5.3 Implementazione del riconoscimento di facce all'interno di NMM

Il processo di riconoscimento delle facce avviene in due passi successivi: per prima cosa, data l'immagine di una faccia in ingresso, ne vengono estratte le feature tramite l'algoritmo Eigenfaces. Successivamente, tali feature vengono classificate utilizzando SVM.

Il modulo che effettua il riconoscimento delle facce attraverso queste due fasi è integrato all'interno del nodo MultiCameraManagerNode, descritto nel paragrafo 6.1. Tuttavia, per la creazione del modello sono stati utilizzati due ulteriori nodi implementati all'interno di NMM:

- `FaceEigensExtractionNode`, per l'estrazione delle feature data un'immagine di una faccia;
- `WriteFeaturesNode`, per la scrittura di tali feature all'interno di un file.

`5.3.1 FaceEigensExtractionNode`

Il nodo `FaceEigensExtractionNode` ha un jack di ingresso e uno di uscita: in ingresso riceve un'immagine contenente una faccia e in uscita restituisce un vettore di double contenente le feature estratte.

Per la creazione della base delle eigenfaces e per la successiva estrazione delle feature sono state utilizzate rispettivamente le funzioni `cvCalcEigenObjects` e `cvEigenDecomposite`,

presenti all'interno della libreria opencv. In particolare, all'interno del costruttore del nodo viene calcolata la base delle eigenfaces, che viene successivamente utilizzata tramite la funzione `cvEigenDecomposite` all'interno del metodo `ProcessBuffer`.

5.3.2 Creazione del modello

La creazione del modello con cui effettuare poi il riconoscimento delle facce viene svolta offline utilizzando l'eseguibile fornito con la libreria `libsvm`. Più nel dettaglio, per creare il modello sono stati realizzati dei video per ciascuna persona da classificare: tali video formano il training set. Ciascuno di tali video è stato fornito in ingresso al nodo `FaceDetectionNode`, allo scopo di isolare la faccia della persona dal resto dell'immagine. La faccia trovata è stata quindi successivamente fornita in ingresso al nodo `FaceEigensExtractionNode`, il quale ne ha estratto le feature che sono state poi inoltrate al nodo `WriteFeaturesNode`, il cui scopo era quello di scrivere le feature estratte all'interno di un file utilizzando il formato richiesto dalla libreria `libsvm`.

Dopo aver completato questa operazione per tutti i video presenti nel training set, e dopo aver scalato le feature tramite il programma `svmscale`, il file ottenuto è stato fornito come parametro al programma `svmtrain`, che ha creato il modello desiderato.

La Figura 5.5 mostra il grafo NMM utilizzato per la creazione del file necessario per generare il modello:

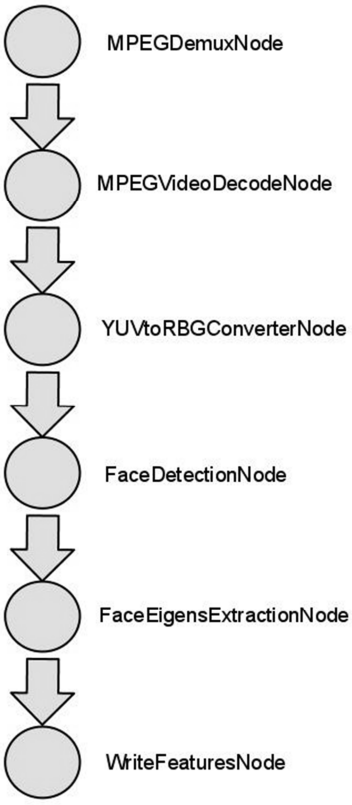


Figura 5.5 - Grafo NMM utilizzato per creare il file necessario per generare il modello.

La Figura 5.6 mostra il dataset che è stato utilizzato per creare il modello; tale dataset è composto da tre persone.

Figura 5.6 – Immagini dal dataset che è stato fornito in ingresso all'applicazione di riconoscimento distribuita per creare il modello.

Infine, in Figura 5.7 è possibile vedere una rappresentazione grafica del training set ottenuta mediante la tecnica della PCA (Principal Component Analysis), in cui si possono notare i tre cluster di punti corrispondenti alle tre persone del dataset:

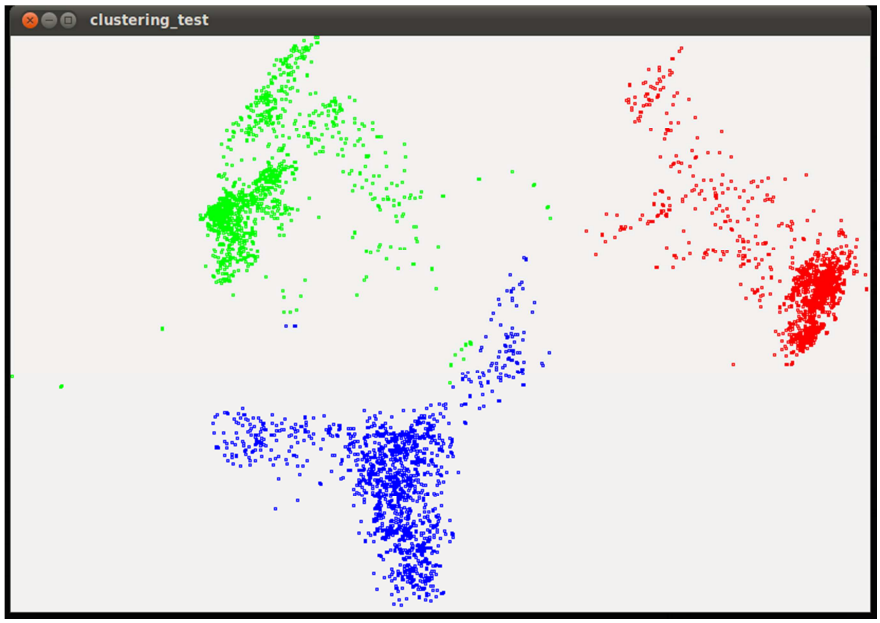


Figura 5.7 - Rappresentazione del dataset utilizzato ottenuta mediante la tecnica delle PCA.

5.4 Test e limiti dell'algoritmo attuale

Dopo aver creato un modello tramite la procedura descritta al paragrafo 5.3.2, si è passati a testare l'algoritmo implementato. In particolare, i test eseguiti e di seguito descritti, hanno avuto lo scopo di verificare i limiti già descritti nel paragrafo 5.1.3.

I test sono stati eseguiti registrando altri video in condizioni prima simili e poi diverse da quelle del training set. Ciascuno di tali video è stato fornito in ingresso al grafo in Figura 5.5, ottenendo il corrispondente file delle feature. Il test è stato eseguito tramite un apposito programma presente nella libreria libsvm che, facendo uso del file delle feature e del modello creato in precedenza, esegue una predizione sui dati in ingresso confrontandone poi il risultato con la label corretta.

Nel dettaglio, sono stati eseguiti test variando le seguenti condizioni:

- distanza dalla telecamera
- inclinazione della testa
- angolazione della testa
- condizioni di luminosità della stanza.

La Figura 5.8 mostra il training set utilizzato. In questo training set sono presenti quattro persone.

Figura 5.8 - Training set utilizzato per il test dell'algoritmo.

5.4.1 Test 1: test set con condizioni uguali a quelle del training set

Il primo test effettuato consisteva nel realizzare un test set avente condizioni uguali a quelle del training set, e rappresentato in Figura 5.9. Come ci si aspettava, si è ottenuta un'accuracy vicina al 100% per tutte le persone coinvolte nel test.

Figura 5.9 - Test set utilizzato per il primo test dell'algoritmo.

5.4.2 Test 2: rotazione della testa

Il secondo test set consisteva in video in cui la testa era sempre più ruotata rispetto alla posizione originale. L'esito di tale test è stato altamente insoddisfacente, rivelando che l'algoritmo attuale risente anche di una leggera rotazione della testa rispetto alla sua posizione nel training set. Infatti, anche con una leggera rotazione della testa come quella del primo video realizzato, l'accuracy dell'algoritmo è scesa bruscamente, portandosi al 12%.

La Figura 5.10 illustra i video utilizzati per il test set:

Figura 5.10 - Test set utilizzato per il secondo test.

5.4.3 Test 3: inclinazione della testa

Il test successivo consisteva nella realizzazione di una serie di video in cui la testa risultasse sempre più inclinata rispetto alla sua posizione nel training set. Anche in questo caso le prestazioni dell'algoritmo si sono rilevate insoddisfacenti, confermandone i limiti descritti dagli autori dello stesso. Infatti, già con una leggera inclinazione della testa come quella del primo video realizzato si è ottenuta un'accuracy pari a circa il 2%.

Figura 5.11 - Test set utilizzato per il terzo test dell'algoritmo.

5.4.4 Test 4: variazione della distanza dalla telecamera

Il quarto test prevedeva la realizzazione di una serie di video in cui la persona variava la propria distanza dalla telecamera. Anche questo test ha confermato il problema della scalatura descritto dagli stessi autori dell'algoritmo e riportato nel paragrafo 5.1.3, ovvero ha dimostrato come le prestazioni dell'algoritmo scendano bruscamente qualora la persona si trovi a distanze dalla telecamera diverse da quella del training set.

Figura 5.12 - Test set utilizzato per il quarto test dell'algoritmo.

5.4.5 Test 5: variazione delle condizioni di luminosità della stanza

L'ultimo test effettuato consisteva nel variare le condizioni di luminosità dell'ambiente. Come i precedenti, anche quest'ultimo test ha dimostrato che l'algoritmo è molto sensibile alle variazioni di luminosità rispetto alle condizioni del training set.

Figura 5.13 - Test set utilizzato per il quinto test dell'algoritmo

5.5 Possibili miglioramenti futuri

Un possibile miglioramento dell'algoritmo si potrebbe ottenere estraendo delle feature più robuste ai cambiamenti di contesto dell'ambiente rispetto a quelle attuali. Tuttavia, anche un elaborazione dell'immagine della faccia prima di effettuare il riconoscimento potrebbe portare a dei miglioramenti. In particolare possibili elaborazioni delle immagini sono:

- Convertire l'immagine in scala di grigi e successivamente applicare un'*histogram equalization* per rendere standard la variazione di luce ed il contrasto dell'immagine. Un esempio di questa elaborazione si può osservare in Figura 5.14:

Figura 5.14 - Elaborazioni aggiuntive precedenti alla fase di riconoscimento - Applicazione dell'*histogram equalization*. (Immagine presa dal sito www.shervinemami.info (23))

- Effettuare un ridimensionamento dell'immagine per portare le immagini delle facce tutte alla stessa dimensione. Tuttavia un semplice ridimensionamento potrebbe modificare il rapporto d'aspetto (*aspect ratio*) dell'immagine. Nel sito http://www.shervinemami.info/imageTransforms.html (23) si può trovare un'utile guida per ridimensionare un'immagine mantenendone invariato il rapporto d'aspetto (si tratta nello specifico di tagliare gli angoli dell'immagine e ingrandirne la parte centrale).

6 Riconoscimento di facce distribuito

Questo capitolo ha lo scopo di descrivere il nodo implementato per la gestione del sistema di riconoscimento distribuito. Verrà infine descritto il grafo completo dell'applicazione realizzata e l'interazione tra i nodi dello stesso.

6.1 MultiCameraManagerNode

Per la gestione dell'applicazione distribuita è stato realizzato il nodo `MultiCameraManagerNode`. Tale nodo svolge diverse funzioni, tra cui le seguenti:

- ricezione delle facce e delle coordinate dalle varie telecamere presenti nel sistema;
- merging dei vari dati all'interno di un oggetto di memorizzazione comune;
- scelta della successiva faccia da riconoscere attraverso le apposite funzioni implementate;
- gestione dello zoom delle eventuali telecamere PTZ presenti all'interno del sistema;
- visualizzazione delle informazioni attraverso una semplice interfaccia grafica.

Il nodo è un nodo demultiplexer; in particolare ha un numero di jack di ingresso variabile a seconda del numero di telecamere utilizzate e del massimo numero di facce inviato da queste ultime. La Figura 6.1 illustra la struttura del nodo nel caso di utilizzo di tre telecamere, tra cui una telecamera PTZ:

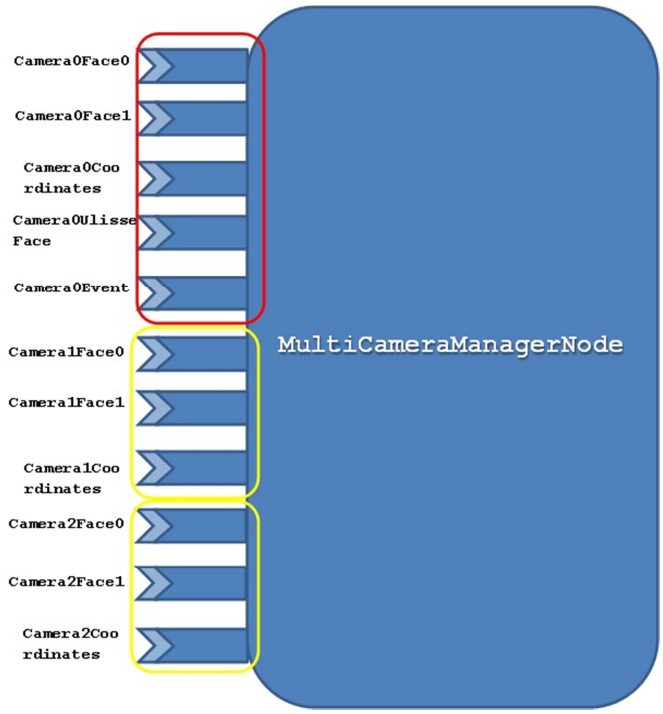


Figura 6.1 - Rappresentazione dei jack di ingresso e di uscita del nodo `MultiCameraManagerNode` nel caso di tre telecamere (tra cui una PTZ) e un massimo di due facce inviabili da ciascuna telecamera.

Come si può osservare dalla Figura 6.1, supponendo che ogni telecamera possa inviare un massimo di due facce ad ogni frame, il nodo presenta tre jack di ingresso per ciascuna telecamera: i primi due jack ricevono le immagini delle facce, mentre il terzo riceve le coordinate delle persone individuate, calcolate come descritto nel paragrafo 4.2.1. Per gestire la presenza di una telecamera PTZ il nodo ha due ulteriori jack di ingresso, "*Camera0UlisseFace*" e "*Camera0Event*": il primo riceve direttamente le facce individuate dalla telecamera nel caso quest'ultima stia zoomando su una persona e il secondo ha lo scopo di inviare e ricevere eventi riguardanti il movimento e il funzionamento della telecamera PTZ stessa.

6.1.1 Oggetti per la memorizzazione dei dati all'interno del nodo

Ogni persona individuata all'interno della stanza viene memorizzata all'interno di un oggetto di tipo `Person`. Questo oggetto contiene le coordinate della persona individuata, un vettore con tutte le facce associate a tale persona, un vettore per memorizzare i risultati del riconoscimento di tali facce e una variabile indicante l'ultimo istante di tempo in cui la persona è stata riconosciuta da una telecamera. Ciascuno di questi oggetti è identificato all'interno del nodo `MultiCameraManagerNode` da un identificatore univoco. Lo schema UML della classe `Person` è rappresentato in Figura 6.2:

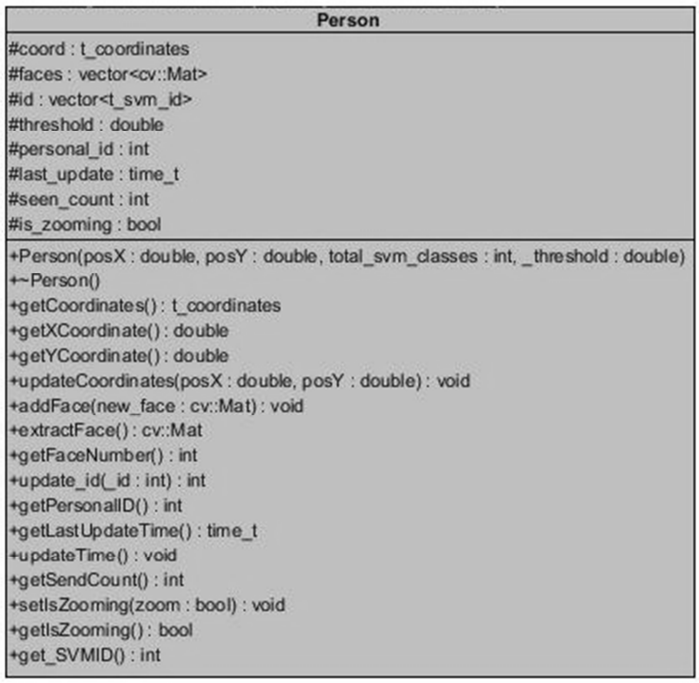


Figura 6.2 - Schema UML della classe `Person`.

I risultati del modulo per il riconoscimento delle facce sono memorizzati all'interno di un vettore di `t_svm_id`; un `t_svm_id` non è altro che una coppia di interi, in cui il primo numero rappresenta una classe del modello SVM, mentre il secondo un contatore che viene

incrementato ogni volta che la persona è riconosciuta appartenere a tale classe. Tale vettore può essere rappresentato anche tramite una tabella, come quella in figura:

SVM Class	Count
0	count0
1	count1
2	count2
3	count3
…	…

Figura 6.3 - Rappresentazione del vettore presente all'interno di ciascun oggetto `Person` per la memorizzazioni dei risultati del modulo per il riconoscimento delle facce.

Ogni volta che il nodo riceve il risultato del riconoscimento di una faccia, invoca il metodo `update_id` dell'oggetto `Person` corrispondente, fornendo in ingresso tale risultato: in questo caso viene incrementato il contatore corrispondente. Allo stesso modo, per ottenere la classe SVM corrispondente ad un oggetto `Person`, viene invocato il metodo `get_SVMID` di quest'ultimo. Ogni volta che tale metodo viene invocato, è applicato il seguente algoritmo al fine di determinare la classe SVM della persona:

- vengono sommati tutti i contatori di tutte le classi (si supponga di indicare questa somma con `sum_count`);
- per ciascuna classe j viene calcolato il valore $\frac{count_j}{sum\_count}$ (dove $count_j$ indica il contatore della classe j);
- una persona viene riconosciuta appartenere alla classe k solamente se

$$\frac{count_k}{sum\_count} > threshold$$

  dove *threshold* è un numero reale strettamente maggiore di 0,5 e minore di 1.

Tuttavia il nodo `MultiCameraManagerNode` non si interfaccia direttamente con i vari oggetti `Person`, bensì con un oggetto di tipo `People`. Tale oggetto non è altro che un vettore di oggetti `Person`, con alcune funzionalità aggiuntive, come la ricerca per coordinate di una persona e la possibilità di cancellare da questa struttura dati le persone che per un certo periodo di tempo non vengono più rilevate dalle telecamere. In Figura 6.4 si può osservare la rappresentazione UML completa della classe `People`:

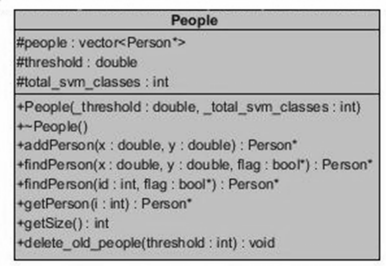


Figura 6.4 - Rappresentazione UML della classe `People`.

6.1.2 Ricezione dei dati dai jack di ingresso

Poiché le coordinate di una persona vengono ricevute separatamente dalla sua faccia, e poiché per memorizzare la faccia di una persona all'interno di un oggetto `Person` sono necessarie le coordinate, non è possibile una memorizzazione immediata all'interno di quest'ultimo dei dati ricevuti. Inoltre, NMM non permette di impostare il jack dal quale si vogliono ricevere i dati: la ricezione dei dati per nodi di tipo multiplexer è gestita internamente dal middleware stesso, che ad ogni esecuzione sceglie il primo tra i vari jack aventi un dato da trasmettere, secondo una politica di scheduling *round robin*. Per questi motivi, il nodo contiene delle strutture temporanee che memorizzano i dati di ingresso ricevuti finché non è possibile trasferirli all'interno di oggetti di tipo `Person`.

Nello specifico, sono stati realizzati due oggetti, `TemporalCoordStorage` e `TemporalFaceStorage`, per la memorizzazione temporanea, rispettivamente, delle coordinate e delle eventuali facce ricevute. In Figura 6.5 e in Figura 6.6 si possono osservare i diagrammi UML di tali classi:

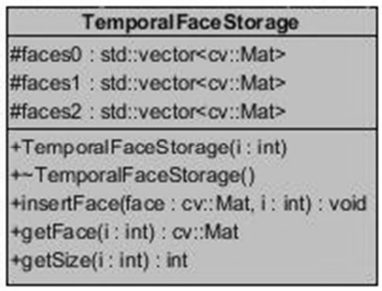


Figura 6.5 - Diagramma UML della classe TemporalFaceStorage

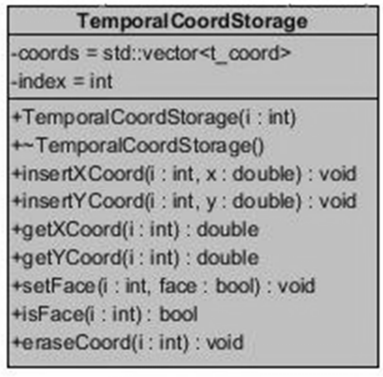


Figura 6.6 - Diagramma UML della classe TemporalCoordStorage

L'unico caso nel quale non è necessaria la memorizzazione temporanea dei dati in ingresso si ha quando vengono ricevute le facce di una persona sulla quale la telecamera PTZ sta zoomando: in questo caso infatti il nodo possiede un puntatore all'oggetto `Person`

corrispondente ed è quindi in grado di memorizzare le facce ricevute direttamente all'interno di quest'ultimo.

Poiché gli algoritmi per il calcolo delle coordinate globali esposti nel capitolo 2 sono soggetti a degli errori, la stessa persona vista da due telecamere diverse potrebbe avere due coordinate leggermente differenti tra loro. Per ovviare a tale problema, le coordinate ricevute vengono quantizzate. Nello specifico, il nodo `MultiCameraManagerNode` divide la stanza in quadrati di un certo lato, e associa a tutte le coordinate appartenenti allo stesso quadrato le stesse coordinate. La Figura 6.7 rappresenta meglio tale quantizzazione:

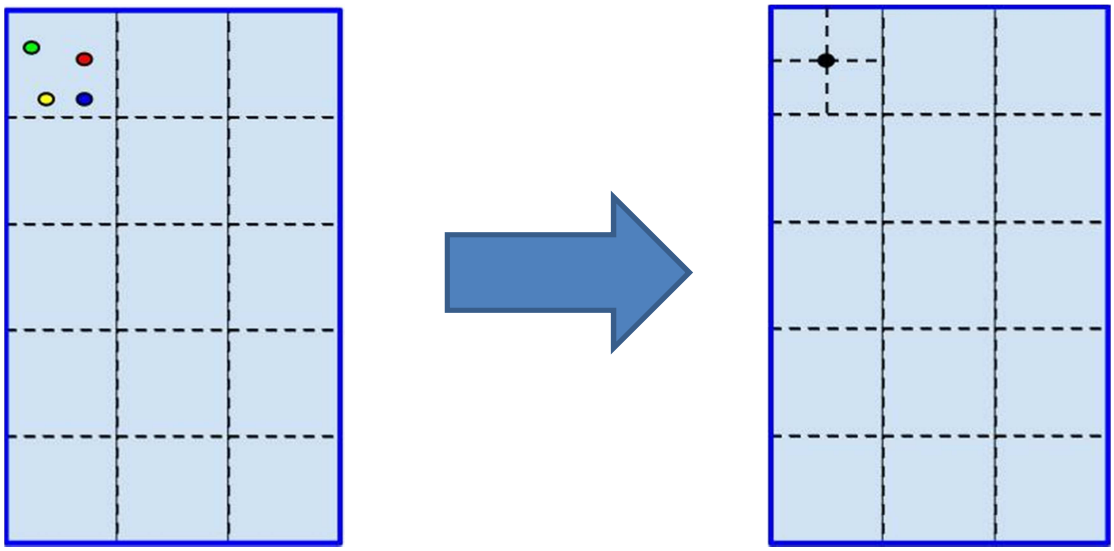


Figura 6.7 - Quantizzazione delle coordinate.

Viene di seguito riportato lo pseudocodice dell'algoritmo completo che gestisce la ricezione dei dati in ingresso e la loro memorizzazione all'interno degli oggetti `Person`:

```
Algorithm ReadInput(Buffer input)
camera_index ← indice della telecamera da cui si sta ricevendo
l'input
current_jack ← nome del jack da cui si sta ricevendo l'input
temp_coord ← vettore di oggetti TemporalCoordStorage
temp_face ← vettore di oggetti TemporalFaceStorage
if (current_jack == "Camera0UlisseFace")
      //la telecamera PTZ sta zoomando e
      //si sta ricevendo una faccia rilevata
      if (la telecamera PTZ sta zoomando)
            controlla se l'immagine in ingresso è completamente nera
            if (l'immagine non è completamente nera)
```

```
                    zooming.addFace()
else
      compared ← "Camera" + camera_index + "Coordinates"
      if (current_jack == compared)
            //si stanno ricevendo delle coordinate
            temp_coord[camera_index].insertXCoord(0, input[0])
            temp_coord[camera_index].insertYCoord(0, input[1])
            temp_coord[camera_index].insertXCoord(1, input[2])
            temp_coord[camera_index].insertYCoord(2, input[3])
            temp_coord[camera_index].insertXCoord(3, input[4])
            temp_coord[camera_index].insertYCoord(4, input[5])
            flag ← input[6]
            temp_coord[camera_index].setFace(0, (flag & 1 == 1))
            temp_coord[camera_index].setFace(1, (flag & 2 == 2))
            temp_coord[camera_index].setFace(2, (flag & 4 == 1))
      else
            //si sta ricevendo una faccia
            face_index ← indice della faccia ricevuta
            controlla se l'immagine ricevuta è completamente nera
            if (l'immagine non è completamente nera)
               temp_face[camera_index].insertFace(input, face_index)
loadPeople(camera_index)
```

```
Algorithm loadPeople(int camera_index)
room_people ← oggetto di tipo People
faceNumber ← massimo numero di facce inviate da una telecamera
if (sono state ricevute le facce e le coordinate da camera_index)
    for (i=0 to faceNumber)
      coord_x ← temp_coord[camera_index].getXCoord(i)
      coord_y ← temp_coord[camera_index].getYCoord(i)
      quantizza le coordinate coord_x e coord_y
      person ← room_people.findPerson(coord_x, coord_y)
      if (found)
            //la persona era già stata rilevata
            person.updateTime()
            if (temp_coord[camera_index].isFace(i))
                  person.addFace(temp_face[camera_index].getFace(i)
      else
            person ← room_people.addPerson(coord_x, coord_y)
```

```
            person.updateTime()
            if (temp_coord[camera_index].isFace(i))
                  person.addFace(temp_face[camera_index].getFace(i)
```

6.1.3 Invio delle facce

Ad ogni esecuzione del metodo `ProcessBuffer`, il nodo `MultiCameraManagerNode` verifica se i vari oggetti `Person` presenti nel nodo contengono delle facce sulle quali si possono applicare le funzioni che ne tentano il riconoscimento. Per la scelta dell'eventuale faccia da riconoscere, il nodo applica uno scheduler di tipo *round robin*, cercando quindi ad ogni iterazione la prima faccia disponibile e preferendo ogni volta una faccia appartenente ad un oggetto `Person` diverso da quello in cui era memorizzata l'ultima faccia scelta.

La Figura 6.8 esemplifica meglio l'approccio *round robin* utilizzato dal nodo `MultiCameraManagerNode`:

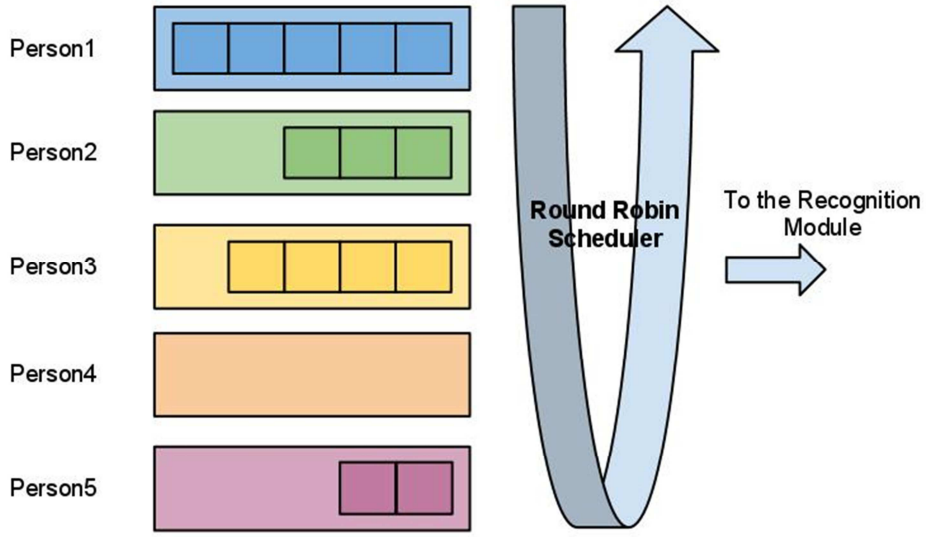


Figura 6.8 - Schema dell'algoritmo di schedulazione *round robin* utilizzato dal nodo `MultiCameraManagerNode` per l'invio delle facce al nodo che ne effettua il riconoscimento.

6.1.4 Modulo per il riconoscimento

Nel costruttore del nodo viene inizializzato il modello per l'estrazione delle feature, ovvero viene creato lo spazio delle facce utilizzando lo Yale DB, attraverso la funzione `cvCalcEigensObject` presente nelle librerie OpenCV.

Una volta scelta la faccia da inviare, essa viene fornita come parametro d'ingresso alla funzione `cvEigenDecomposite`, ottenendo in uscita il vettore contenente i valori delle varie feature. Tale vettore viene poi passato come parametro di input alla funzione `svm_classifier`, implementata all'interno del nodo stesso.

La classificazione implementata è composta principalmente da tre fasi:

- scalatura delle feature ricevute;
- invocazione della funzione presente all'interno della libreria `libsvm` per ottenere la predizione della classe e le varie confidenze;
- interpretazione di tali confidenze e predizione della classe finale.

Tutte le feature vengono scalate all'interno dell'intervallo $[-1,1]$. In particolare, siano M_i e m_i rispettivamente il massimo e il minimo valore assunti dalla i-esima feature; scalare tale feature all'interno dell'intervallo $[-1,1]$ significa applicare la seguente equazione:

$$x_{scaled} = \frac{2 \cdot (x - m_i)}{(M_i - m_i)} - 1$$

dove *x* è il valore non scalato della feature.

Successivamente, viene invocata la funzione `svm_predict_values` implementata all'interno della libreria `libsvm`, che restituisce come parametri di uscita la predizione effettuata dalla libreria e le varie confidenze.

Le confidenze restituite vengono quindi analizzate una prima volta al fine di verificare se sono presenti due classi che hanno vinto entrambe lo stesso numero di confronti, pari al massimo numero di confronti vinti in totale. In questo caso, infatti, `libsvm` restituisce come vincitore la prima di queste classi; nel modulo di riconoscimento implementato è stato invece scelto di restituire come vincitore la classe avente il valore di confidenza più grande in valore assoluto.

Una volta determinata la classe predetta, viene applicata una soglia a tutte le confidenze che coinvolgono quest'ultima: in particolare, viene restituita tale classe come risultato finale del riconoscimento solo se è presente almeno una confidenza maggiore della soglia scelta. Nel caso questo non si verifichi, viene restituita una classe indicante che il riconoscimento non è riuscito ad identificare la faccia, catalogandola quindi come sconosciuta.

6.1.5 Interfaccia grafica

All'interno del nodo `MultiCameraManagerNode` è stata sviluppata una semplice interfaccia grafica che, utilizzando le librerie opencv, visualizza una mappa della stanza e le varie persone che si trovano al suo interno. In particolare, nell'interfaccia grafica vengono indicate in rosso le persone individuate ma non ancora riconosciute dal sistema, in blu con il loro nome le persone riconosciute e in verde la persona sulla quale il sistema sta zoomando con la telecamera PTZ.

Poiché il risultato del modulo di riconoscimento è un numero intero che rappresenta la classe SVM, il nodo `MultiCameraManagerNode` contiene al suo interno un oggetto di tipo `SVMTranslator` che converte tale intero in una stringa rappresentante il nome della persona, che viene poi visualizzata nell'interfaccia grafica.

La Figura 6.9 mostra l'interfaccia grafica del sistema:

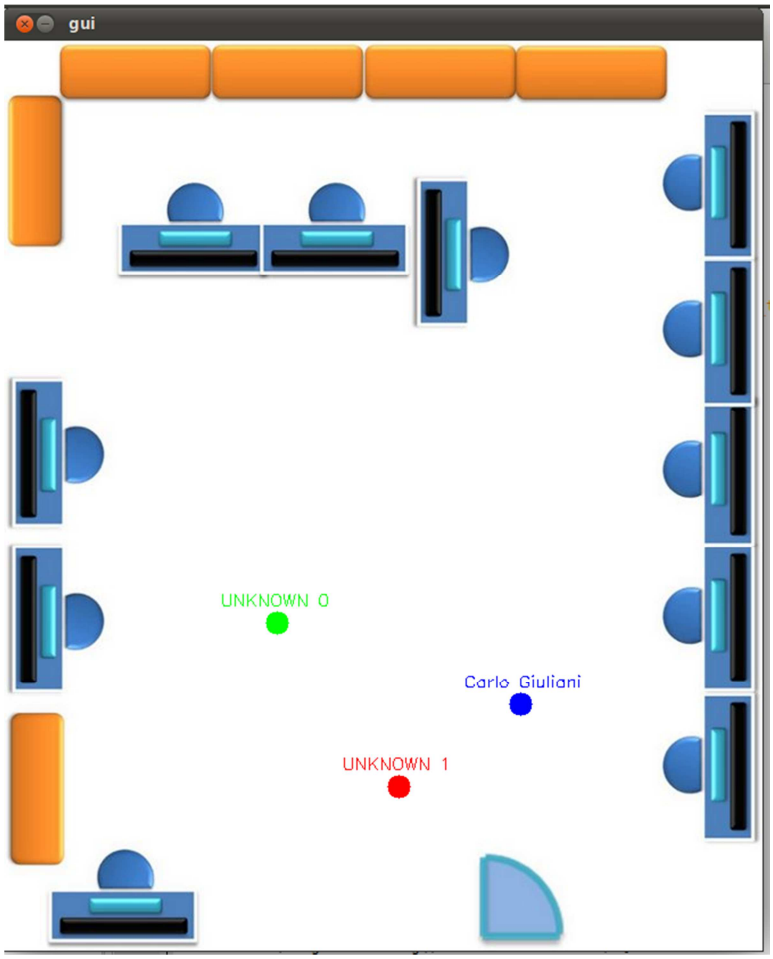


Figura 6.9 - Interfaccia grafica del nodo `MultiCameraManagerNode`.

6.1.6 Gestione dello zoom delle telecamere PTZ

Una caratteristica importante del nodo `MultiCameraManagerNode` è la gestione dello zoom delle telecamere PTZ collegate al sistema. In particolare, all'interno del nodo è presente un algoritmo che verifica la presenza di persone non ancora riconosciute all'interno della stanza e inoltra alle varie telecamere PTZ le loro coordinate, affinché esse possano zoomare sulla faccia di queste ultime. Più specificatamente, ad ogni esecuzione del metodo `ProcessBuffer` viene chiamata la funzione `zoomOnFace`, che effettua una scansione lineare di tutti gli oggetti `Person` presenti all'interno del nodo cercandone uno che rappresenta una persona non ancora riconosciuta dal sistema. Qualora tale oggetto venga trovato, la funzione recupera le coordinate della persona e le inserisce all'interno di un evento, che viene poi inviato in direzione *upstream* all'interno del grafo, facendo in modo che le telecamere PTZ zoomino nel punto desiderato.

6.2 Descrizione dell'applicazione completa

In Figura 6.10 viene riportato il grafo completo dell'applicazione che effettua il riconoscimento di facce nel sistema di sensori distribuito.

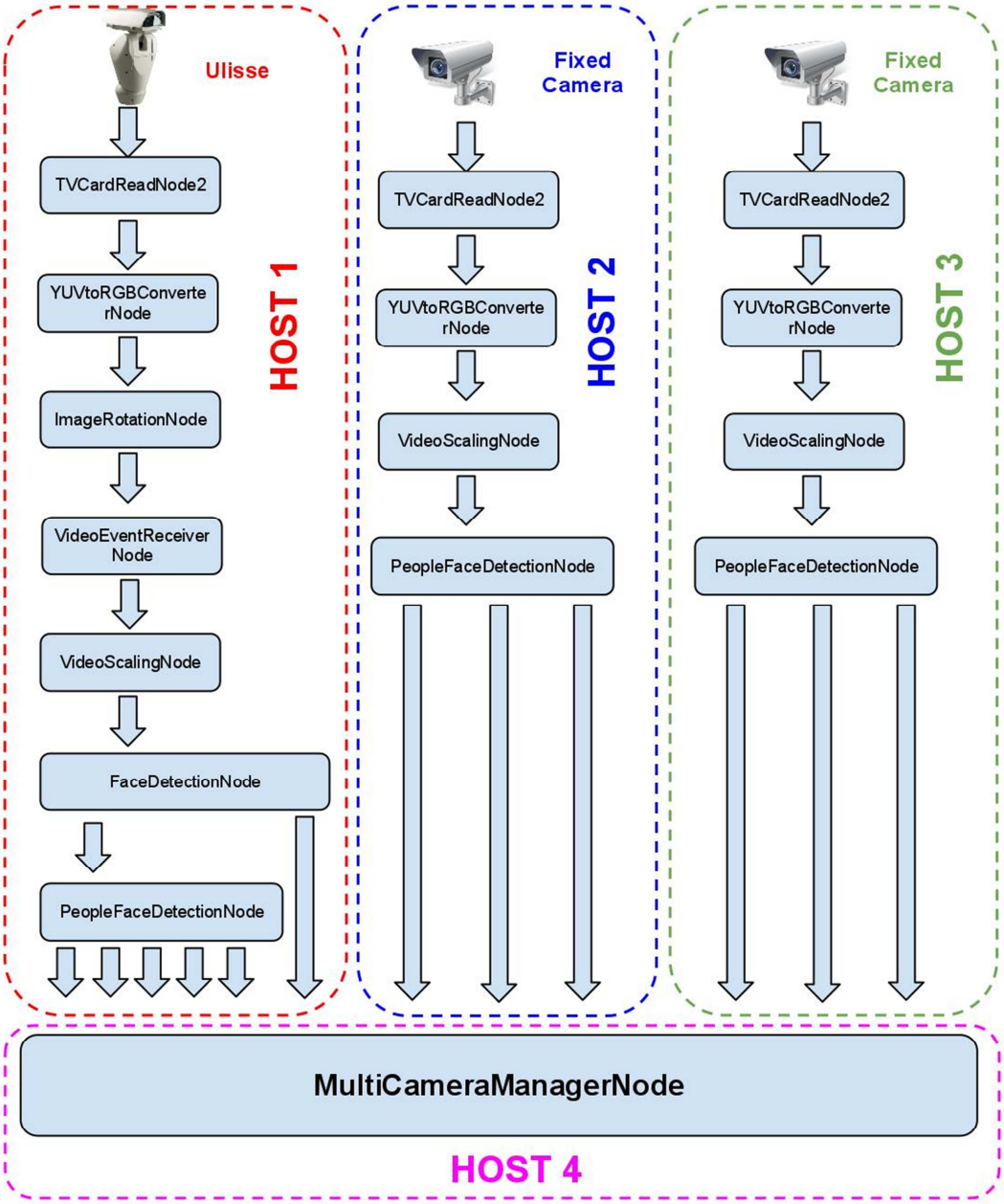


Figura 6.10 - Rappresentazione del grafo completo dell'applicazione che effettua il riconoscimento di facce distribuito.

I nodi presenti nell'applicazione e non ancora descritti nel corso di questa tesi sono i seguenti:

- `TVCardReadNode2`: è un nodo di tipo source avente lo scopo di leggere l'input dal frame grabber al quale è collegata la telecamera;
- `YUVtoRGBConverterNode`: è un nodo che converte lo stream d'ingresso dalla codifica YUV alla codifica RGB;
- `ImageRotationNode`: è un nodo che ruota lo stream d'ingresso di 180°;
- `VideoScalingNode`: scala l'immagine in ingresso portandola alla risoluzione 640x480.

- `VideoEventReceiverNode`: gestisce il movimento della telecamera PTZ Ulisse.

6.2.1 Invio degli eventi che gestiscono lo zoom della telecamera PTZ Ulisse

L'algoritmo completo che gestisce lo zoom della telecamera Ulisse coinvolge più nodi all'interno del grafo; in particolare sono coinvolti i nodi `MultiCameraManagerNode`, `PeopleFaceDetectionNode`, `FaceDetectionNode` e `VideoEventReceiverNode`.

L'algoritmo che gestisce lo zoom della telecamera PTZ può idealmente essere suddiviso in due fasi, la prima avente lo scopo di iniziare a zoomare sulla faccia di una persona e la seconda a terminare la fase di zoom.

Per quanto riguarda l'inizio della fase di zoom sulla faccia di una persona, viene applicata la seguente procedura:

1. Il nodo `MultiCameraManagerNode` verifica che siano soddisfatte le seguenti condizioni:
   - l'Ulisse non sta già zoomando su una persona;
   - vi sono persone all'interno della stanza;
   - è passato dall'ultimo zoom un tempo superiore ad un tempo fissato come parametro all'interno del nodo;
   - non è ancora stato inviato un evento per zoomare sulla faccia di una persona.

   Nel caso tutte le condizioni precedenti siano soddisfatte, il nodo `MultiCameraManagerNode` cerca se, all'interno della stanza, vi sono persone non ancora riconosciute; se almeno una di queste ultime è presente, viene generato un evento *ZOOM_ON_FACE* contenente le coordinate della persona nel sistema di riferimento globale. L'evento viene poi inviato nel grafo in direzione *upstream*.
2. Il nodo `PeopleFaceDetectionNode` è in ascolto per l'evento *ZOOM_ON_FACE*. Quando uno di questi ultimi eventi viene ricevuto, il nodo calcola gli angoli di pan e tilt necessari per centrare la faccia della persona nel piano immagine della telecamera utilizzando l'algoritmo descritto nel paragrafo 2.4.2. Viene poi calcolato anche lo zoom necessario basandosi su misurazioni effettuate sperimentalmente (più specificatamente, si è misurato in modo sperimentale lo zoom necessario per inquadrare la faccia di una persona posta ad una certa distanza dalla telecamera e, in base a tale misurazione, viene calcolata prima la distanza della persona dalla telecamera e poi lo zoom necessario).

   Il nodo `PeopleFaceDetectionNode` genera quindi un evento *ULISSE_ZOOM* contenente gli angoli di pan, di tilt e lo zoom e lo invia in direzione *upstream* nel grafo.
3. Il nodo `VideoEventReceiverNode` è in ascolto per l'evento *ULISSE_ZOOM*. Quanto tale evento viene ricevuto, il nodo invia all'Ulisse il comando necessario per posizionarsi nella posizione ricevuta. Successivamente, il nodo crea due eventi,

ULISSE_IS_ZOOMING e *ULISSE_IS_ZOOMING_P*, contenenti l'attuale posizione dell'Ulisse, e invia questi ultimi nel grafo in direzione *downstream*.

4. Il nodo `FaceDetectionNode` è in ascolto per l'evento *ULISSE_IS_ZOOMING*. Quando quest'ultimo viene ricevuto, il nodo aggiorna la sua variabile interna indicante il fatto che l'Ulisse sta zoomando su una persona; inoltre il nodo viene attivato e comincia ad effettuare il rilevamento delle facce sullo stream video ricevuto in ingresso. Le eventuali facce rilevate vengono direttamente inviate al nodo `MultiCameraManagerNode`.
5. Il nodo `PeopleFaceDetectionNode` è in ascolto per l'evento *ULISSE_IS_ZOOMING_P*. Qualora tale evento venga ricevuto, il nodo aggiorna la sua variabile interna indicante il fatto che l'Ulisse sta zoomando sulla faccia di una persona e ferma il rilevamento delle persone sullo stream video ricevuto in ingresso, ponendosi in una condizione di "stand-by", in cui si limita ad inoltrare al nodo `MultiCameraManagerNode` immagini contenenti solamente uno sfondo nero.

   Inoltre, il nodo `PeopleFaceDetectionNode` genera un evento *ULISSE_IS_ZOOMING_M* contente gli angoli di pan, tilt e lo zoom attuale dell'Ulisse, inviando poi tale evento in direzione *downstream* nel grafo.
6. Infine, il nodo `MultiCameraManagerNode` è in ascolto per l'evento *ULISSE_IS_ZOOMING_M*. Una volta ricevuto tale evento, il nodo aggiorna la propria variabile interna che contiene il timestamp dell'istante in cui l'Ulisse ha iniziato a zoomare su una persona (che l'algoritmo assume essere uguale all'istante in cui l'evento *ULISSE_IS_ZOOMING_M* viene ricevuto); inoltre viene aggiornata la variabile interna che tiene traccia dello stato dello zoom dell'Ulisse. Tale variabile serve ad indicare al metodo `ProcessBuffer` che può accettare le eventuali immagini di facce provenienti dal nodo `FaceDetectionNode`.

La Figura 6.11 illustra graficamente la sequenza degli eventi inviati al fine di iniziare a zoomare sulla faccia di una persona:

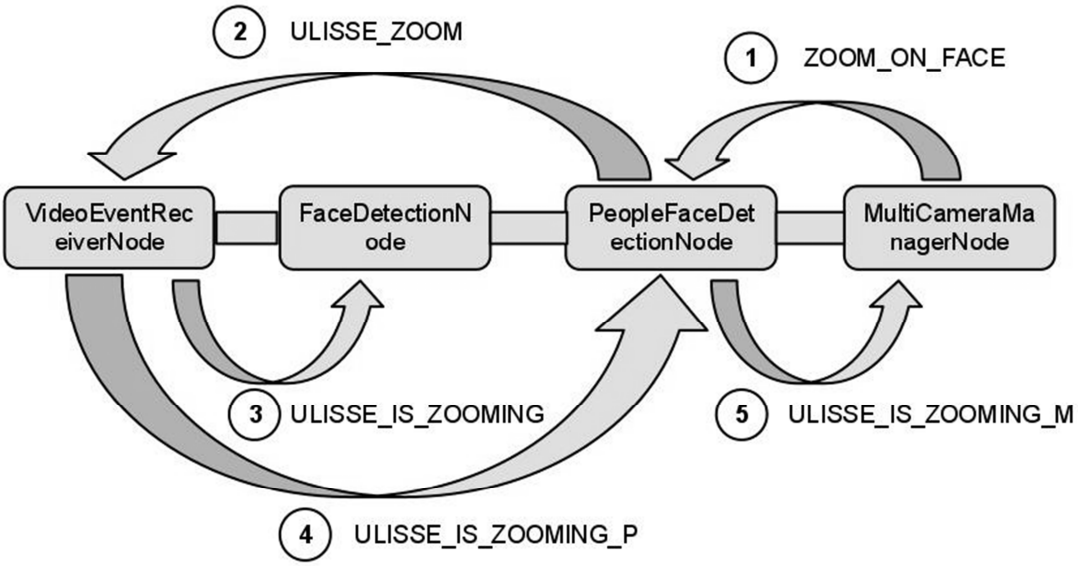


Figura 6.11 - Sequenza degli eventi inviati per iniziare a zoomare sulla faccia di una persona.

Per terminare la fase di zoom sulla faccia di una persona viene seguita la seguente procedura:

1. Il nodo `MultiCameraManagerNode` verifica che siano soddisfatte le seguenti condizioni:
   - l'Ulisse sta zoomando su una persona;
   - non è ancora stato inviato l'evento per indicare la fine della fase di zoom;
   - l'Ulisse sta zoomando per un tempo maggiore della massima durata di una fase di zoom.

   Se tutte le condizioni precedenti sono soddisfatte, il nodo genera un evento *STOP_ZOOM* e invia quest'ultimo in direzione *upstream* nel grafo.
2. Il nodo `VideoEventReceiverNode` è in ascolto per l'evento *STOP_ZOOM*. Una volta ricevuto tale evento, il nodo invia un comando di riposizionamento all'Ulisse lasciando invariati gli angoli di pan e di tilt, ma reimpostando a 1 lo zoom della telecamera. Successivamente il nodo crea due eventi, *ULISSE_IS_NOT_ZOOMING* e *ULISSE_IS_NOT_ZOOMING_P*, e invia tali eventi nel grafo in direzione *downstream*.
3. Il nodo `FaceDetectionNode` è in ascolto per l'evento *ULISSE_IS_NOT_ZOOMING*. Quando tale evento viene ricevuto, il nodo aggiorna la sua variabile interna indicante lo stato dello zoom della telecamera e termina di effettuare la rilevazione delle facce sul flusso video in ingresso, tornando ad inoltrare semplicemente quest'ultimo sul suo jack di uscita collegato al nodo `PeopleFaceDetectionNode`.

4. Il nodo `PeopleFaceDetectionNode` è in ascolto per l'evento *ULISSE_IS_NOT_ZOOMING_P*. Una volta ricevuto tale evento, il nodo aggiorna la proprie variabili interne indicanti lo stato dello zoom dell'Ulisse e gli attuali angoli di pan e di tilt della telecamera stessa. Il nodo riavvia poi la rilevazione delle persone nel flusso video in ingresso e genera l'evento *ULISSE_IS_NOT_ZOOMING_M*, che viene inviato nel grafo in direzione *downstream*.
5. Il nodo `MultiCameraManagerNode` è in ascolto per l'evento *ULISSE_IS_NOT_ZOOMING_M*. Alla ricezione di quest'ultimo, il nodo aggiorna le proprie variabili interne indicanti lo stato dello zoom dell'Ulisse e il timestamp dell'ultimo istante in cui è terminata una fase di zooming, che l'algoritmo assume essere uguale all'istante in cui viene ricevuto l'evento *ULISSE_IS_NOT_ZOOMING_M*.

La Figura 6.12 illustra graficamente la sequenza dell'invio degli eventi per terminare la fase di zoom sulla faccia di una persona:

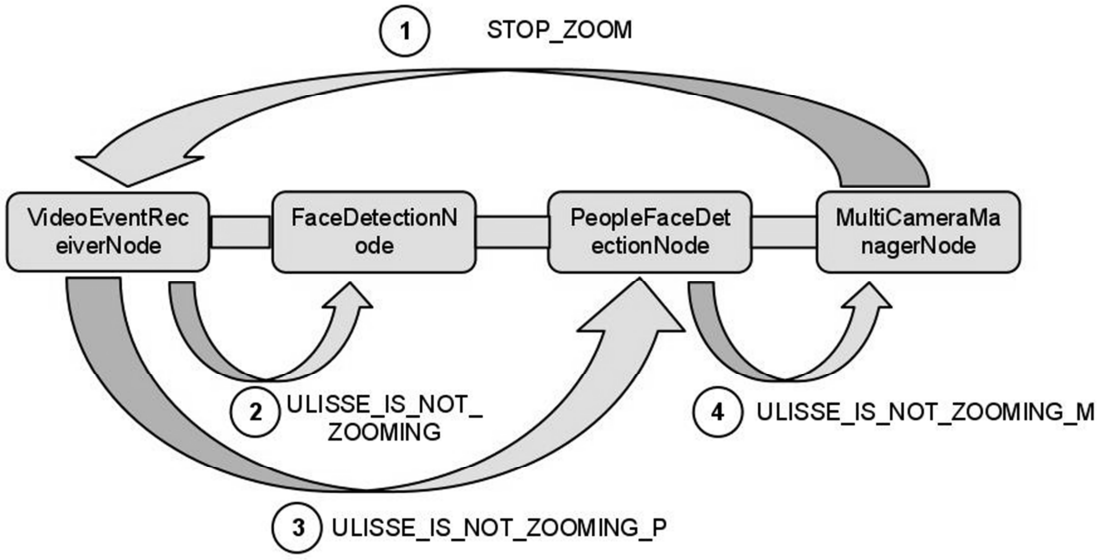


Figura 6.12 - Sequenza degli eventi inviati per terminare la fase di zoom sulla faccia di una persona.

Come si può osservare dalla Figura 6.11 e dalla Figura 6.12, gli eventi *ULISSE_IS_ ZOOMING_M* e *ULISSE_IS_NOT_ZOOMING_M* sono generati dal nodo `PeopleFaceDetectionNode` e non dal nodo `VideoEventReceiverNode`: questo è semplicemente dovuto ad un bug di NMM, per il quale non è possibile inviare più di due eventi da uno stesso nodo senza causare la terminazione immediata dell'applicazione.

Infine, le figure seguenti mostrano degli screenshot dell'applicazione in esecuzione. In particolare, la Figura 6.13 mostra l'applicazione appena avviata che ha rilevato la presenza di due persone all'interno del laboratorio. La Figura 6.14 mostra l'applicazione mentre sta

zoomando sulla faccia di una delle due persone, che viene anche riconosciuta dal sistema; la Figura 6.15 mostra la telecamera PTZ che ha terminato la fase di zoom e sta applicando l'algoritmo di rilevamento delle persone al flusso video in ingresso. Infine, la Figura 6.16 mostra la telecamera PTZ che sta zoomando sulla faccia della seconda persona rilevata, che viene anch'essa riconosciuta dal sistema.

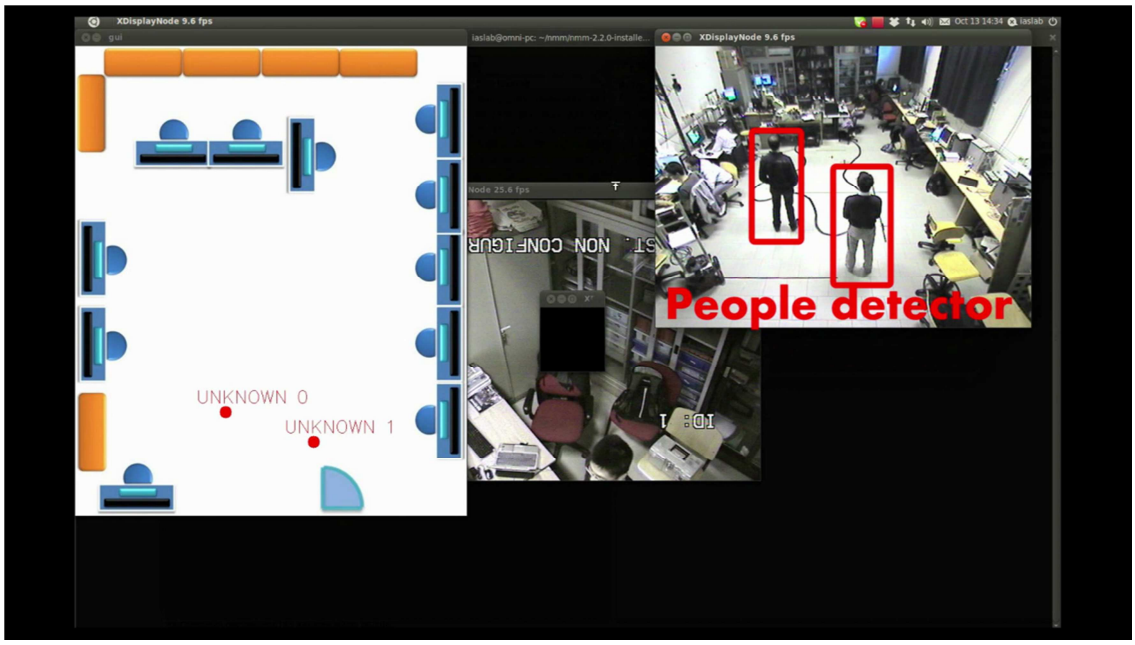


Figura 6.13 - Screenshot dell'applicazione. Si può notare la telecamera fissa che ha rilevato la presenza di due persone all'interno della stanza e la telecamera PTZ Ulisse che sta inquadrando un'altra area del laboratorio.

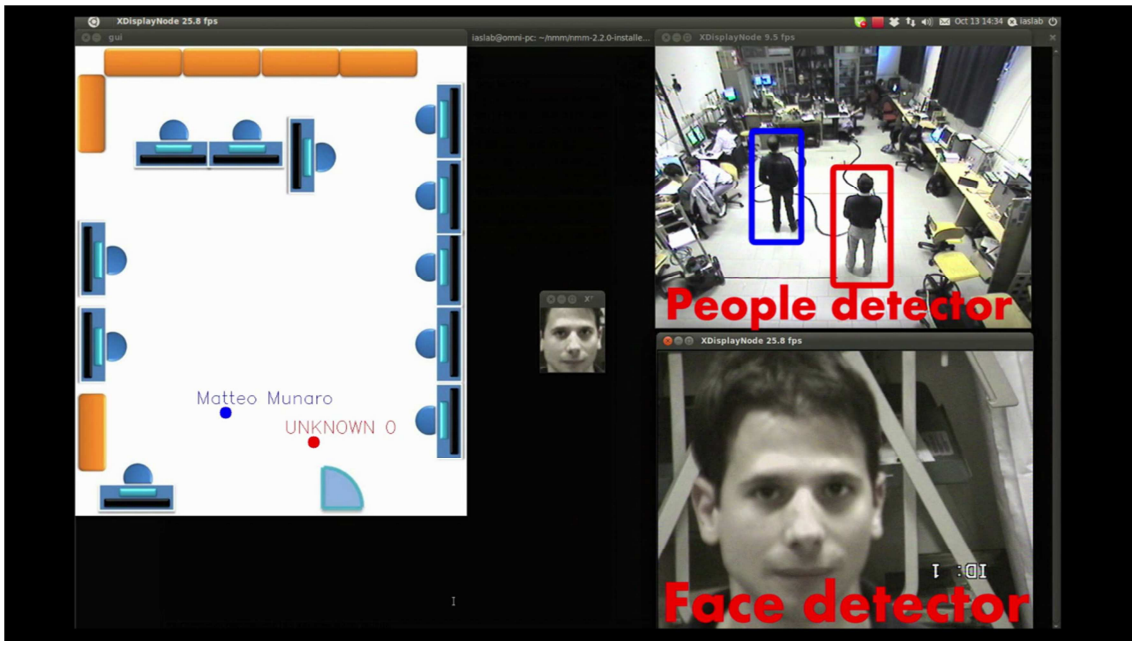


Figura 6.14 - Screenshot dell'applicazione. La telecamera PTZ Ulisse ha modificato i suoi angoli di pan e di tilt e lo zoom inquadrando la faccia della prima persona rilevata. Quest'ultima è stata poi riconosciuta dal modulo di riconoscimento.

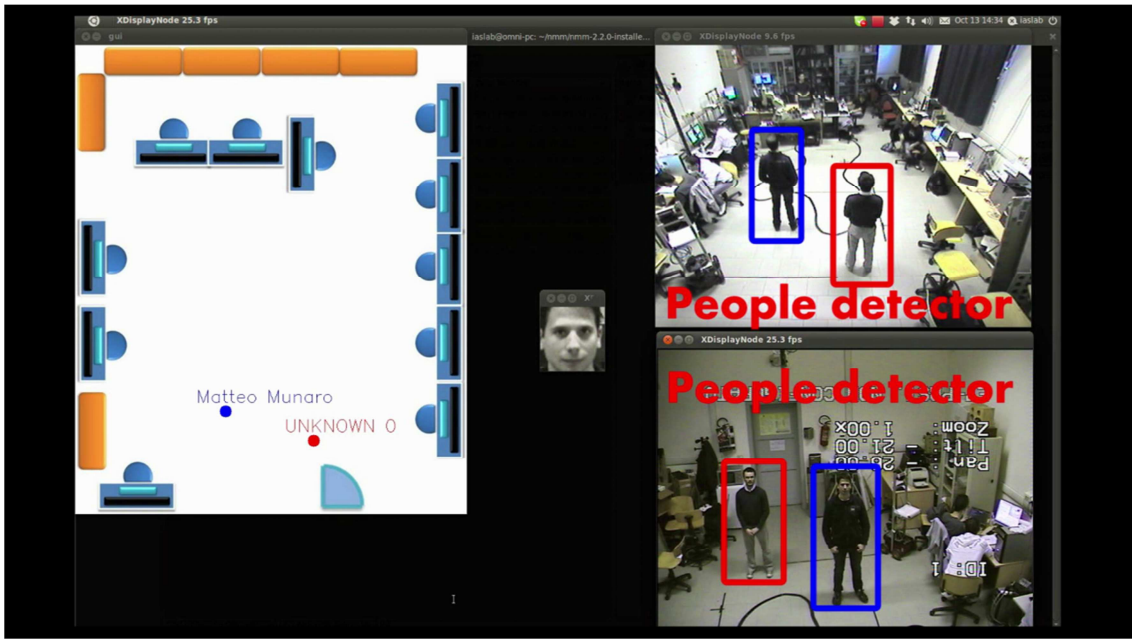


Figura 6.15 - Screenshot dell'applicazione. La telecamera PTZ ha smesso di zoomare sulla faccia della persona ed è tornata ad applicare l'algoritmo per il rilevamento delle persone al flusso video in ingresso.

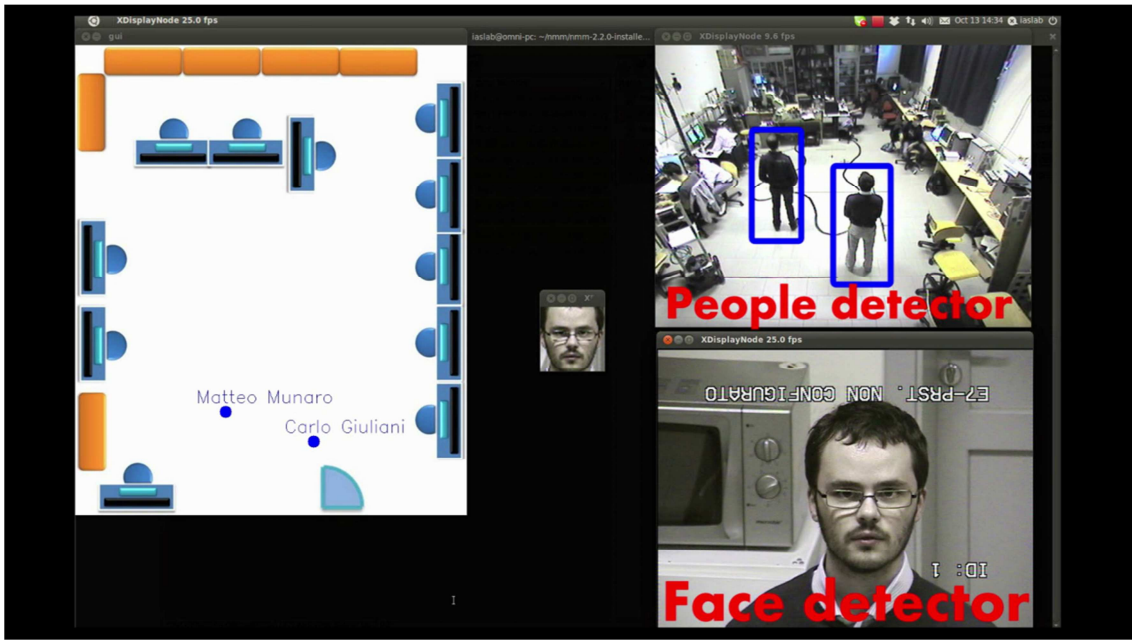


Figura 6.16 - Screenshot dell'applicazione. La telecamera PTZ ha zoomato sulla faccia della seconda persona presente all'interno della stanza. Quest'ultima è stata poi riconosciuta dal modulo di riconoscimento.

6.3 Risultati

Sono stati infine realizzati dei test al fine di misurare le performance in termini temporali dell'applicazione realizzata. In particolare, sono state effettuate quattro misurazioni:

- tempo necessario per effettuare il rilevamento delle persone all'interno di un flusso video;
- tempo necessario per effettuare il rilevamento delle facce all'interno della bounding box ottenuta dal rilevamento delle persone;

- tempo necessario per effettuare il rilevamento delle facce all'interno dell'interno flusso video;
- tempo necessario per effettuare la predizione dei dati con SVM.

Le misurazioni sono state effettuate utilizzando un PC avente un processore Intel® Xeon® E31225 a 3,10 GHz e 4 GB di memoria RAM; il sistema operativo utilizzato per eseguire i test è Ubuntu 11.04. I risultati delle misurazioni sono riportati in Tabella 6.1 (i risultati sono riportati in microsecondi):

	People Detection (μs)	**Face Detection nel bounding box (μs)**	**Face Detection (μs)**	**Recognition (μs)**
	77902	6669	167338	182
	82151	5599	183076	137
	91912	13322	189186	103
	87050	10942	202648	104
	87215	9543	191743	173
	82408	9543	154538	100
	86742	8618	168552	129
	83035	8656	178404	91
	84784	8609	184106	149
	81241	14768	186773	151
Media	**84444**	**9627**	**180636**	**132**

Tabella 6.1 - Risultati del test dell'applicazione realizzata.

Pertanto, il rilevamento delle persone ha un frame rate di circa 12 frame/s, il rilevamento delle facce a partire dalla bounding box di una persona ha un frame rate di 103 frame/s, il rilevamento delle facce a partire dall'intero flusso video ha un frame rate di 5,5 frame/s e il riconoscimento attraverso SVM ha un frame rate di 7575 frame/s.

Il frame rate complessivo dell'applicazione è determinato dal frame rate più basso, ed è quindi di 12 frame/s quando tutte le telecamere sono in fase di rilevazione delle persone e nessuna sta zoomando, per scendere a 5,5 frame/s nel momento in cui la telecamera PTZ sta zoomando su una persona.

7 Conclusioni e sviluppi futuri

Questo lavoro di tesi si è incentrato principalmente sulla creazione di un modello di riferimento comune a tutti i sensori utilizzati e sull'implementazione di un nodo che fosse in grado di gestire stream video provenienti dalle diverse telecamere collegate a vari host all'interno di una rete locale.

Il principale obiettivo raggiunto è stato quindi la creazione di un'infrastruttura in grado di fondere informazioni provenienti da diverse istanze dell'applicazione per il rilevamento delle facce presenti all'interno della rete locale e che comunichi allo stesso tempo con un modulo centralizzato in grado di effettuare il riconoscimento di queste ultime.

Innanzitutto, sono stati realizzati degli algoritmi che consentono di creare un modello di riferimento comune a tutte le telecamere presenti in laboratorio. La principale novità introdotta in questo senso è l'integrazione di tale modello con la presenza di una telecamera PTZ, che può variare i suoi angoli di pan e di tilt e quindi anche il suo piano immagine. In questo senso, si è riusciti ad elaborare un algoritmo che presenta un tasso di errore relativamente basso e che può quindi essere utilizzato per altre applicazioni che coinvolgano una rete di sensori.

È stato poi realizzato un nodo che gestisce la comunicazione tra i vari sensori presenti all'interno della stanza ed il processo di riconoscimento. Tale nodo, oltre a consentire la distribuzione del software di rilevazione e riconoscimento delle persone, presenta dei vantaggi rispetto al semplice utilizzo in sequenza delle librerie OpenCV per l'estrazione delle feature e di libSVM per il loro riconoscimento. In particolare, l'applicazione realizzata tiene in considerazione anche i valori delle varie confidenze ottenute tramite SVM, facendo una predizione solo qualora tali valori siano al di sopra di una certa soglia. Inoltre, si è in grado di mantenere una statistica dei vari riconoscimenti, limitando in questo modo eventuali errori dovuti ad una errata classificazione delle feature.

Infine, l'utilizzo del nodo sviluppato congiuntamente ad una rete di sensori consente di gestire in maniera efficiente informazioni provenienti da questi ultimi e di riconoscere persone presenti all'interno della stanza anche qualora esse non siano visibili pienamente da alcuni di essi.

L'applicazione ottenuta ha un frame rate complessivo di 5,5 frame/s. Questo risultato implica che, affinché l'applicazione possa essere utilizzata nell'ambito della videosorveglianza intelligente, è necessario scartare alcuni frame anziché processarli tutti: infatti la videosorveglianza richiede applicazioni che siano in grado di funzionare in tempo reale, e che abbiano quindi un frame rate complessivo attorno ai 25 frame/s. In questo caso si è verificato che l'applicazione realizzata è in grado di funzionare in tempo reale se viene elaborato un frame ogni 5.

Eventuali sviluppi futuri di tale lavoro sono i seguenti:

- Utilizzo di un altro middleware per gestire la distribuzione dell'applicazione: come accennato anche nel capitolo 3, NMM presenta attualmente alcuni difetti che ne rendono l'utilizzo abbastanza oneroso; sarebbe quindi da valutare l'utilizzo di un middleware alternativo, come ROS, che attualmente fornisce una maggiore stabilità.
- Estrazione di feature più resistenti alle variazioni di luce, alla scalatura e ai cambiamenti della posa rispetto alle eigenfaces.
- Utilizzo di una base per la creazione dello spazio delle facce contenente le facce delle persone usualmente presenti all'interno del laboratorio: in questo lavoro di tesi si è utilizzato lo Yale DB in quanto l'obiettivo ultimo del progetto è la creazione di un sistema di riconoscimento che sia in grado di riconoscere qualsiasi persona, e non solo determinate persone presenti solitamente all'interno dell'ambiente. L'uso dello Yale DB consente infatti la creazione di uno spazio delle feature più generale e quindi più predisposto alla proiezione di feature estratte dalla faccia di una persona qualsiasi. Tuttavia, l'utilizzo di un altro database di facce per la creazione della base dello spazio delle facce porterebbe ad un miglioramento delle performance dell'algoritmo di riconoscimento in caso di utilizzo di un dataset chiuso. Infatti, utilizzando come facce per creare la base dello spazio vettoriale proprio le facce delle persone abitualmente presenti all'interno dell'ambiente, si riuscirebbe a creare uno spazio ottimizzato proprio per la proiezione di tali facce, aumentando in questo modo le performance dell'algoritmo quando si tratta di riconoscere tali persone.
- Integrazione del sistema con un algoritmo di tracking, in modo da tener traccia anche del movimento delle varie persone all'interno della stanza e allo stesso tempo mantenere le informazioni legate sulla loro identità.
- Integrazione del sistema con un robot mobile dotato di una telecamera ad alta definizione, che sia in grado di avvicinarsi alle persone non ancora riconosciute dalla rete di sensori per tentare il loro riconoscimento.

8 Bibliografia

1. *Localization and Recognition of Smart Objects by a Mobile Robot.* **Menegatti, E.; Danieletto, M.; Mina, A.; Pretto, A.; Bardella, A.; Zanella, A.; Zanuttigh, P.** Padova : Simulation, Modeling, and Programming for Autonomous Robots, SIMPAR 2010, 2010.

2. *Autonomous discovery, localization and recognition of smart objects through WSN and image features.* **Menegatti, E.; Danieletto, M.; Mina, M.; Pretto, A.; Bardella, A.; Zanconato, S.; Zanuttigh, P.; Zanella, A.** s.l. : International Workshop Towards Smart Communication and Network technologies applied on Autonomous Systems, IEEE GLOBECOM, 2010.

3. *Identification of Human Faces.* **Goldstein, A. J., Harmon, L. D. e Lesk, A. B.** 5, s.l. : Proc. IEEE, 1971, Vol. 59.

4. *A Low-Dimensional Procedure for the Characterization of Human Faces.* **Sirovich, L. e Kirby, M.** 3, s.l. : J. Optical Soc. Am. A, 1987, Vol. 4.

5. *Eigenfaces for Recognition.* **Turk, Matthew e Pentland, Alex.** 1, Massachussetts Institute of Technology : Journal of Cognitive Neuroscience, 1992, Vol. 3.

6. *Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection.* **Belhumeur, Peter N., Hespanha, João P. e Kriegman, David J.** 7, s.l. : IEEE Transactions on pattern analysis and machine intelligence, 1997, Vol. 19.

7. *The Use of Multiple Measurements in Taxonomic Problems.* **Fisher, Ronald Aylmer.** s.l. : Annals of Eugenics, 1936.

8. *SURF-Face: Face Recognition Under Viewpoint Consistency Constraints.* **Dreuw, Philippe, et al., et al.** Aachen, Germany : BMVC, 2009.

9. *Speeded-Up Robust Features (SURF).* **Bay, Herbert, et al., et al.** 3, s.l. : Computer Vision and image understanting, 2008, Vol. 110.

10. *Three-Dimensional Face Recognition.* **Bronstein, Alexander M., Bronstein, Michael M. e Kimmel, Ron.** 1, Haifa, Israel : International Journal of Computer Vision, 2005, Vol. 64.

11. *Distributed Face Recognition via Consensus on SE.* **Tron, Roberto e Vidal, René.** Marseille, France : The 8th Workshop on Omnidirectional Vision, Camera Networks and Non-Classical Cameras, 2008.

12. *Distributed Face Recognition: a Multi-Agent Approach.* **Chetty, Girija e Dharmendra, Sharma.** 8A, University of Canberra, Australia : IJCSNS International Journal of Computer Science and Network Security, 2006, Vol. 6.

13. *Multi-Camera Face Recognition by Reliability-Based Selection.* **Xie, Binglong, et al., et al.** Alexandria, VA, USA : IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety, 2006.

14. *Fusing Face Recognition from Multiple Cameras.* **Harguess, Josh, Hu, Changbo e Aggarwal, J. K.** University of Texas, Austin, TX, USA : 2009 Workshop on Applications of Computer Vision (WACV), 2010.

15. **Bradsky, Gary e Kaehler, Adrian.** *Learning OpenVC - Computer Vision with the OpenCV Library.* s.l. : O'REILLY, 2008.

16. *Indoor PTZ Camera Calibration with concurrent PT Axes.* **Sanchez-Riera, Jordi, Salvador, Jordi e Casas, Josep R.** Barcelona : Proc. of the Fourth Inter. Conf. on Computer Vision Theory and Applications (VISAPP), 2009.

17. Motama GmbH. [Online] http://www.motama.com.

18. *Robust Real-Time Object Detection.* **Viola, Paul e Michael, Jones.** Vancouver, Canada : Second international workshop on statistical and computational theories of vision - modelling, learning, computing and sampling, 2001.

19. *A decision-theoretic generalization of on-line learning and an application to boosting.* **Freund, Yoav e Schapire, Robert E.** s.l. : AT&T Bell Laboratories, 1995.

20. *A Tutorial on Support Vector Machines for Pattern Recognition.* **Burges, Christopher J.C.** s.l. : Data Mining and Knownledge Discovery - Springer, 1998, Vol. 2.

21. Yale Face Database. [Online] http://cvc.yale.edu/projects/yalefaces/yalefaces.html.

22. **Chang, Chih-Chung e Lin, Chih-Jen.** LIBSVM -- A Library for Support Vector Machines. [Online] http://www.csie.ntu.edu.tw/~cjlin/libsvm/.

23. **Emami, Shervin.** Face Detection and Face Recognition with Real-Time training from a camera. [Online] http://www.shervinemami.info/faceRecognition.html.

24. **Laganière, Robert.** *OpenCV 2 Computer Vision Application Programming Cookbook.* s.l. : Packt Publishing, 2011.

25. **Hartley, Richard e Zisserman, Andrew.** *Multiple View Geometry in Computer Vision - Second Edition.* s.l. : Cambridge University Press, 2003.

26. **Lohse, Marco.** *Network-Integrated Multimedia Middleware, Services, and Applications.* Saarbrücken, Germany : Universität des Saarlandes, 2005.

27. *Audio-video people recognition system for an intelligent environment.* **Anzalone, Salvatore M.; Menegatti, Emanuele; Pagello, Enrico; Yoshikawa, Yuichiro; Ishiguro, Hiroshi; Chella, Antonio.**