POLITECNICO DI MILANO
Corso di Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

E-2?: UN NUOVO SISTEMA DI VISIONE PER ROBOT AUTONOMO DI ATTRAZIONE PER AMBIENTI PUBBLICI

AI & R LAB
LABORATORIO DI INTELLIGENZA ARTIFICIALE
E ROBOTICA DEL POLITECNICO DI MILANO

Relatore: Prof. Andrea Bonarini

Tesi di Laurea di:
Giulio Fiscella, matricola 714769
Federico Sem, matricola 715124

Anno Accademico 2009-2010

Sommario

L'obiettivo che si propone questa tesi è lo sviluppo di un algoritmo di face-tracking da utilizzare all'interno del già esistente progetto di E2-?. E2-? nasce come un sistema di intrattenimento robotico per utilizzo in ambienti pubblici. Sviluppato sulla base del progetto Robocup, la finalità del sistema è la creazione di un robot con elevate caratteristiche di autonomia e mobilità, e con comportamenti sufficientemente elaborati da consentirgli un livello di interazione avanzato rispetto agli attuali standard dell'industria. Una buona parte dei processi di interazione operati da E-2? è basata sul riconoscimento di determinate caratteristiche (nel nostro caso, i volti) all'interno dei frame catturati da una videocamera frontale, che costituisce l'apparato di visione del robot. L'algoritmo sviluppato ha permesso di ottenere un notevole miglioramento nelle capacità di interpretazione delle immagini da parte di E-2?. In particolare, si è ottenuto un sistema in grado non solo di riconoscere i volti delle persone con accuratezza, ma anche di stimarne la posizione e l'orientamento nelle tre dimensioni spaziali. La rinnovata capacità di analisi delle immagini del robot ha, infine, permesso lo sviluppo di alcuni schemi comportamentali atti ad evidenziare le migliorie introdotte ed a limare alcuni atteggiamenti indesiderati. I test delle funzionalità, pur essendo stati effettuati in ambienti parzialmente controllati, sono risultati coerenti con gli obiettivi inizialmente preposti.

Ringraziamenti

Indice

Capitolo 1

Introduzione

L'obiettivo che si propone questa tesi è lo sviluppo di un algoritmo di face-tracking da utilizzare all'interno del già esistente progetto di E-2?. E-2? nasce come un sistema di intrattenimento robotico per utilizzo in ambienti pubblici. Sviluppato sulla base del progetto Robocup, la finalità del sistema è la creazione di un robot con elevate caratteristiche di autonomia e mobilità, e con comportamenti sufficientemente elaborati da consentirgli un livello di interazione avanzato rispetto agli attuali standard dell'industria.

Una buona parte dei processi di interazione operati da E-2? è basata sul riconoscimento di determinate caratteristiche (nel nostro caso, i volti) all'interno dei frame catturati da una videocamera frontale, che simula la visione del robot. E' quindi evidente come lo sviluppo di un sistema di rilevazione dei volti sia un'aspetto critico per un robot che fa della capacità di interazione e di relazione con altri esseri umani una delle sue caratteristiche salienti.

Consci dell'importanza ricoperta dal sistema di riconoscimento nel caso di E-2?, il nostro scopo è stato l'ottenimento di un sistema di rilevamento che permettesse al robot di comportarsi in maniera naturale e il più possibile prevedibile all'interno di ambienti non controllati. L'approccio adottato è consistito in una riscrittura totale del codice relativo al rilevatore di volti, e nel successivo interfacciamento dell'algoritmo con il software di controllo del robot.

In una seconda fase si è provveduto a creare un insieme di comportamenti basati sui dati che il software è in grado di fornire, e studiati per migliorare l'interazione uomo-robot. Principalmente, ci si è concentrati sul miglioramento del comportamento di avvicinamento del robot al proprio target (una faccia umana nel caso specifico) e sullo sviluppo e successivo utilizzo di nuovi tipi di dati che caratterizzano maggiormente il soggetto inquadrato (ad esempio: angoli per

definire l'orientamento e grandezza della testa).

Il lavoro svolto è stato dunque suddiviso in due differenti blocchi concettuali: all'interno del primo blocco si illustra come lo sfuttamento di algoritmi già noti nel settore, opportunamente rielaborati ed adattati al contesto di utilizzo, abbiano portato alla sintesi di un face-tracker robusto e affidabile. Verranno innanzitutto trattate le problematiche relative all'individuazione di features facciali all'interno di un'immagine, e successivamente si sposterà l'attenzione sui metodi di tracciamento che garantiscono maggiori prestazioni e la precisione più elevata. Le specifiche del nuovo algoritmo dovranno essere:

- individuzione istantanea di volti in uno stream video
- tracciamento efficiente dei volti nello spazio
- robustezza, in particolare rispetto a variazioni di orientamento nel soggetto
- utilizzabilità real time

Nella seconda parte, ci si concentrerà maggiormente sulle dinamiche di creazione di nuovi comportamenti a partire dai dati rilevati da un flusso video. In particolare, si mostrerà come utilizzare la logica fuzzy per la definizione di variabili, e si descriveranno i passaggi necessari per utilizzare tali variabili all'interno di predicati logici struttabili per definire il comportamento del robot.

La struttura di massima della tesi è schematizzabile come segue:

- Capitolo 2: viene presentata una descrizione schematica del robot; si analizza innanzitutto la componentistica hardware, partendo dal calcolatore e analizzando successivamente i sensori di interesse per la tesi (sonar e camera). Si passa poi alla descrizione del middleware, ponendo particolare attenzione al suo ruolo come scambiatore di messaggi. Infine, si descrive brevemente il software Mr.Brian per la gestione dei comportamenti.
- Capitoli 3-4: si affrontano innanzitutto gli aspetti matematici alla base degli algoritmi utilizzati; partendo dal modello di Viola e Jones per la rilevazione di features facciali, si descrivono i passaggi per ottenere un sistema di face detection completo, in cui la parte di tracking viene gestita tramite l'implementazione di Optical Flow. La parte finale si occupa di descrivere i passaggi effettuati per implementare questi algoritmi e il loro utilizzo.
- Capitoli 5-6: vengono descritti, in maniera semplice ed esaustiva, i concetti alla base della logica fuzzy, e viene spiegato come il suo utilizzo possa essere

utile al fine di modellizzare i dati in ingresso al robot. Anche in questo caso, alla descrizione matematica segue una sezione interamente dedicata agli aspetti implementativi, riguardanti in questo caso la creazione di dati fuzzy e la gestione dei predicati che definiscono nuovi comportamenti del robot.

- Capitolo 7: Risultati e considerazioni conclusive.

Capitolo 2

Il Robot

Come già detto, E-2? è un progetto del Politecnico di Milano iniziato nel luglio del 2009. Il robot è già costrutito e funzionante e si basa su software e hardware precedentemente creati nell'ambito del progetto MRT (Modular Robot Toolkit) CIT. In questa sezione vengono descritte tutte le componenti che compongono il robot con maggiore riguardo a quelle con cui ci siamo dovuti confrontare nello sviluppo del nostro progetto.

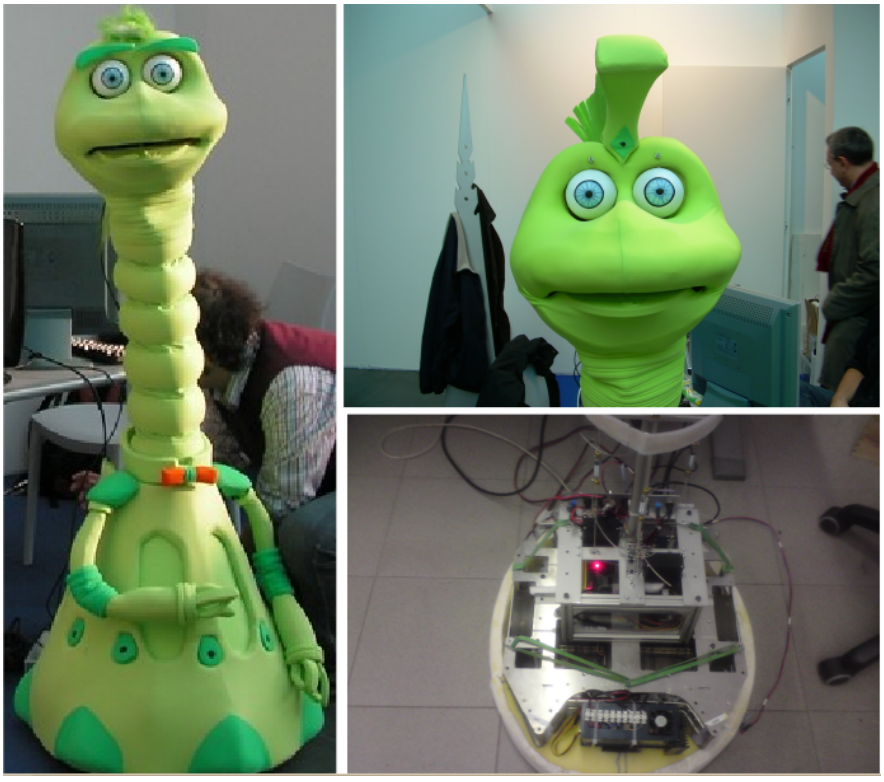


Figura 2.1: Alcune foto del robot e della sua struttura interna

2.1 Hardware

La configurazione hardware di partenza è basata su un precedente progetto sviluppato dal Politecnico di Milano chiamato Milan Robocup Team CIT. In particolare, la struttura hardware di base del progetto è stata realizzata partendo dal robot calciatore chiamato "IANUS". Esso incorpora due ruote nella base, ognuna alimentata da un motore che funziona a 12V. Le ruote sono allineate al centro del corpo.

2.1.1 Calcolatore

Il cuore del robot è costituito da un personal computer PCBrick con scheda madre Mini-ITX, equipaggiata con un processore Core2 Duo T7200 @2GHz, 2 GB di RAM, un HardDisk da 80 GByte di dimensioni standard 2.5 pollici. La piattaforma garantisce elevate prestazioni computazionali, senza sacrificare troppo spazio o consumi.

Figura 2.2: Il PCBrick

2.1.2 Sonar

I sonar offrono diversi tipi di output, nel caso si è optato per l'uscita PWM, in cui l'ampiezza di emissione è proporzionale alla distanza misurata. La cintura sonar, sviluppata ed assemblata in AIRLab, è realizzata con 7 sensori Maxbotix

Ez-2, collegati ad un unico micro controllore. Il micro controllore si occupa di sequenziare i sonar e di convertire le distanze misurate dai singoli sensori (al massimo 4 contemporaneamente) e di inviarle al PC.

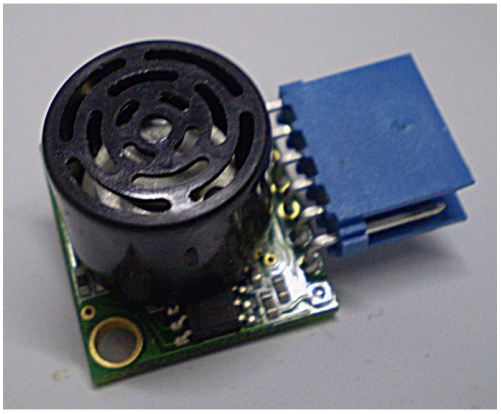


Figura 2.3: Uno dei sonar

Per la rilevazione degli ostacoli nella zona operativa i sonar sono così disposti:

- 3 nella zona frontale che coprono 120°(2,3,4)
- 4 che coprono i 240°rimanenti (0,1,5,6)

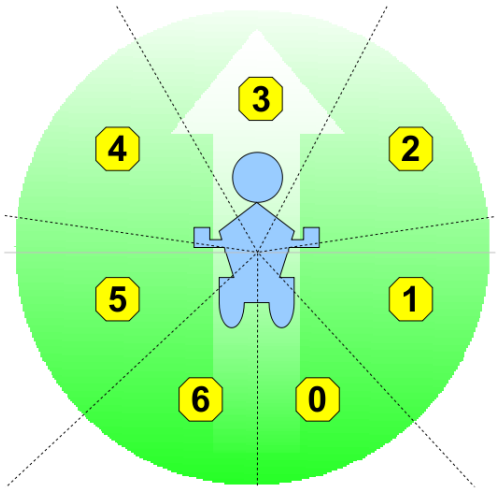


Figura 2.4: Disposizione dei sonar

Tale apparato, oltre che per la rilevazione di ostacoli, viene utilizzato in cooperazione con il modulo di visione per identificare la distanza del target da raggiungere.

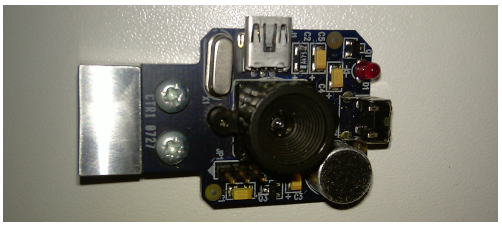
2.1.3 Camera

Inizialmente, era stata utilizzata una micro camera, del tipo incorporabile in notebook o smartphone. Le dimensioni ridotte, dell'ottica in particolare, ne facilitavano l'occultamento ma comportavano una qualità del video insoddisfacente e, soprattutto, scarsa adattabilità ai cambiamenti di luce.

Si è allora deciso di utilizzare una comune webcam USB esterna, con risoluzione massima 640x480, con ottiche migliori e, nonostante ciò, ugualmente occulatabile al di sotto del rivestimento del robot.

La camera, inoltre, incorpora un microfono che sarà sicuramente utile per sviluppi futuri del comportamento.

(a) *La micro camera iniziale.*



(b) *La nuova camera USB.*

Figura 2.5: Le camere

2.1.4 Altri componenti

Nel robot, inoltre, sono presenti 8 sensori ad infrarossi utilizzati per identificare sul pavimento delle striscie pre-posizionate che indichino la zona in cui operare .

Il sistema dei servomotori che muovono testa e robot si articola attorno ad una scheda realizzata ad hoc in AIRLab, in grado di comandare fino a 12 servomotori da modellismo. Il controllo avviene solamente in posizione, ma la scheda è in grado di limitare sia velocità che accelerazione, interpolando i setpoint da dare ai motori.

L'alimentazione è assicurata da 8 batterie SLA (Sealed Lead Acid), batterie sigillate al piombo, con tensione e capacità nominali di 12V e 7.2Ah. L'autonomia misurata è maggiore di 5h di funzionamento continuo.

Infine, abbiamo un dispositivo di controllo a distanza, con una portata di trasmissione/ricezione di circa 10 metri, che permette di passare da modalità autonoma a contralotta e, in quest'ultima, di comandare il robot.

2.2 Middleware

Il DCDT (Device Communities Developement Toolkit), sviluppato all'interno dell'AIRLab e parte integrante del progetto MRT, riesce a incorporare i vari strumenti disponibili e ne garantisce la loro interazione.

Si tratta di un sistema orientato alla gestione degli agenti. Un agente è un'entità astratta rappresentata nel codice da classi di tipo Expert. Per svolgere l'attività a cui sono destinati, gli agenti hanno bisogno di eseguire i loro compiti in totale autonomia e di poter scambiare messaggi con gli altri agenti operanti nel medesimo ambiente. DCDT rende disponibile un ambiente multiagente senza gravare l'utente degli aspetti legati alla programmazione in multithreading e della comunicazione tra i diversi thread.

Per una guida completa a DCDT si rimanda al suo manuale utente.

2.2.1 Messaggi

Lo scambio dei messaggi è rappresentabile con un modello di un ipotetico ufficio postale che ha il compito di ricevere, smistare e consegnare i messaggi mediante una classe chiamata Dispatcher che svolge il lavoro di postino. DCDT ha una politica pubblica/sottoscrivi che nasconde la distribuzione fisica dei messaggi. In questo modello, tutti gli agenti hanno la possibilità di pubblicare messaggi inviandoli all'ufficio postale.

Ad ogni messaggio è assegnato un tipo e un agente ha la possibilità di sottoscriversi ad una lista di interesse, specificando il tipo di messaggi che intende ricevere. In questo modo solo i diretti interessati possono ottenere le informazioni precedentemente richieste e non vengono, quindi, prodotte copie inutili dello stesso messaggio, aumentando l'efficienza e l'affidabilità del sistema.

2.3 Software

La parte software è quindi suddivisa in agenti autonomi in grado di interagire tra loro. L'agente di maggior interesse (oltre ovviamente all'agente coordinatore: *Brian*) per la realizzazione del sistema di face-tracking è chiamato *VisionFrontalExpert*. Gli agenti che forniscono un ruolo di supporto sono *SonarExpert* (rilevazione di distanza) e *MotorExpert* (movimento verso il soggetto). Di seguito uno schema di massima:

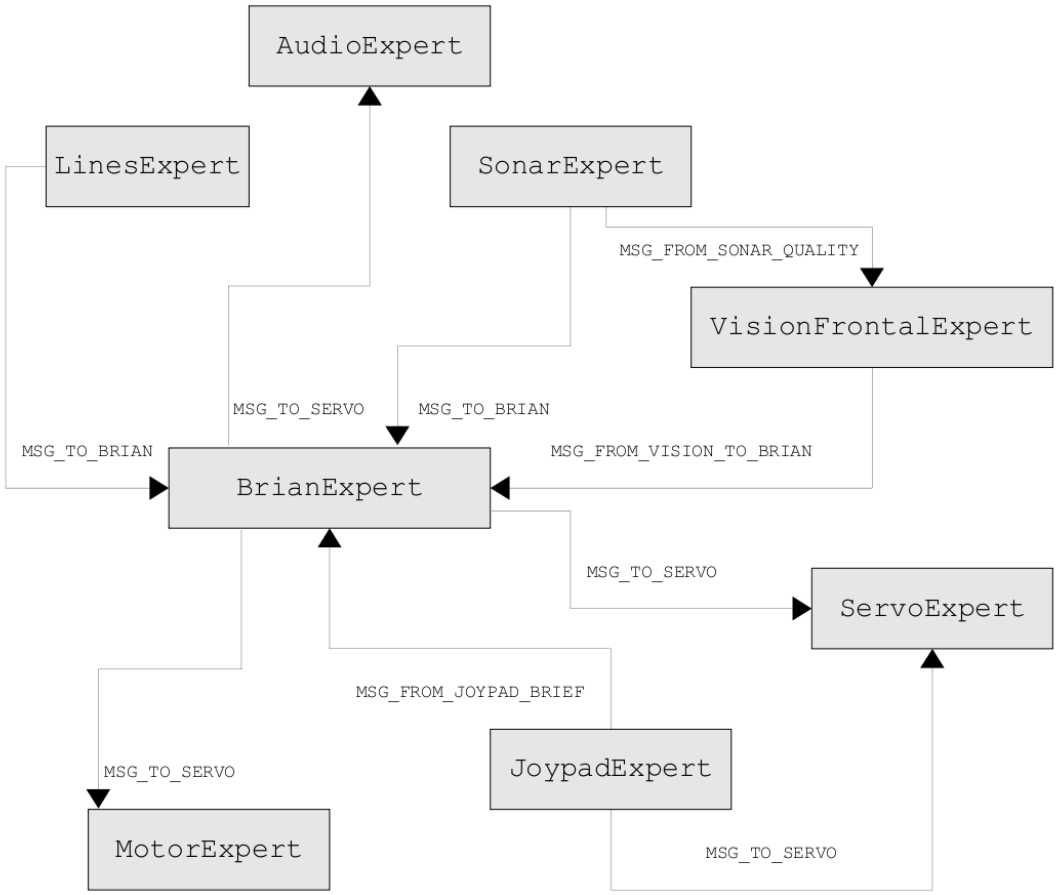


Figura 2.6: Schema a blocchi dello scambio di messaggi tra gli agenti

Negli esperti che gestiscono i sensori la fase di creazione del messaggio può essere scomposta in tre passaggi:

- Acquisizione dei dati dal sensore
- Aggiunta degli attributi al messaggio
- Invio del messaggio a Mr.Brian

2.3.1 Mr. Brian

Mr.Brian (Multilevel Ruling BRIAN) CIT, è un estensione di BRIAN (BRIAN Reacts by Inferring ActioNs) software sviluppato dal Politecnico di Milano sempre all'interno del progetto MRT. Esso permette di utilizzare dei predicati espressi in logica fuzzy (si veda il capitolo relativo per una introduzione) per fornire le

regole di comportamento che il robot impiegherà per svolgere i propri compiti in modo autonomo.

Tralasciando, inizialmente, gli aspetti legati alle gerarchie fra i vari comportamenti e riferendoci in particolare al software Brian, il processo che, a partire dai dati di input, produce in output l'azione di controllo è il seguente:

1. Fuzzyficazione dei dati di ingresso
2. Valutazione del valore di verità dei predicati
3. Scelta dei comportamenti da attivare
4. Valutazione delle regole dei singoli comportamenti
5. Fusione dei risultati
6. Defuzzyficazione dei risultati

Capitolo 3

L'Algoritmo

Il problema standard di face detection, data un'immagine qualsiasi, può essere definito nel seguente modo: determinare ogni viso (se presente) all'interno dell'immagine, fornendo per ciascuno posizione e dimensioni. Idealmente, l'intera procedura deve essere effettuata indipendentemente dalle variazioni di illuminazione, scala e orientamento del soggetto. Dunque, la robustezza è l'aspetto principale da tenere in considerazione in un qualunque sistema di rilevamento.

I metodi di Face Detection possono essere classificati secondo numerosi criteri. Uno dei più usati considera l'informazione utilizzata per modellizzare i volti come discrimine per la classificazione delle differenti tecniche:

- Implicita o Pattern based: si tratta di approcci che funzionano cercando un pattern precedentemente memorizzato in ogni porzione di immagine e per differenti scale della stessa.

- Esplicita o Knowledge Based: sono metodi che aumentano la velocità di esecuzione assumendo che l'algoritmo conosca esplicitamente i volti da analizzare ed effettui il rilevamento combinando colore, movimenti e/o geometria facciale.

Gli algoritmi appartenenti alla seconda famiglia si contraddistinguono per la rapidità di esecuzione ma solo in determinate condizioni. I primi, invece, affrontano il problema generale del face detection nelle immagini statiche, ottenendo ottimi risultati in termini di errore e falsi positivi. Se applicati, però, direttamente a sequenze di immagini non considerano l'informazione implicita nell'evoluzione temporale, scartando le informazioni derivanti dalle rilevazioni precedenti, quali posizione, dimensioni e aspetto del viso.

Di conseguenza, si è deciso di utilizzare un approccio che fa uso di elementi appartenenti ad entrambe le categorie nel tentativo di sfruttarne i vantaggi: performance ottime della prima famiglia e velocità della seconda.

Iniziamo, quindi, a descrivere l'algoritmo da noi proposto, di cui si può trovare una schematizzazione in alcune delle figure che seguono. Il funzionamento può essere suddiviso in due modalità principali, a seconda dell'avvenuto rilevamento di volti nella storia recente.

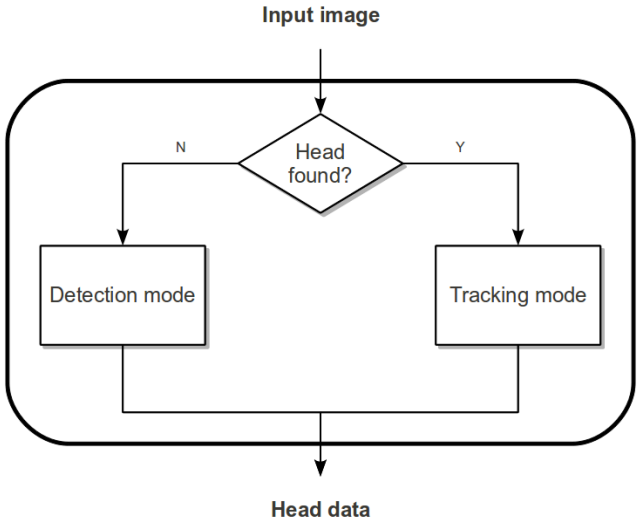


Figura 3.1: Schema base

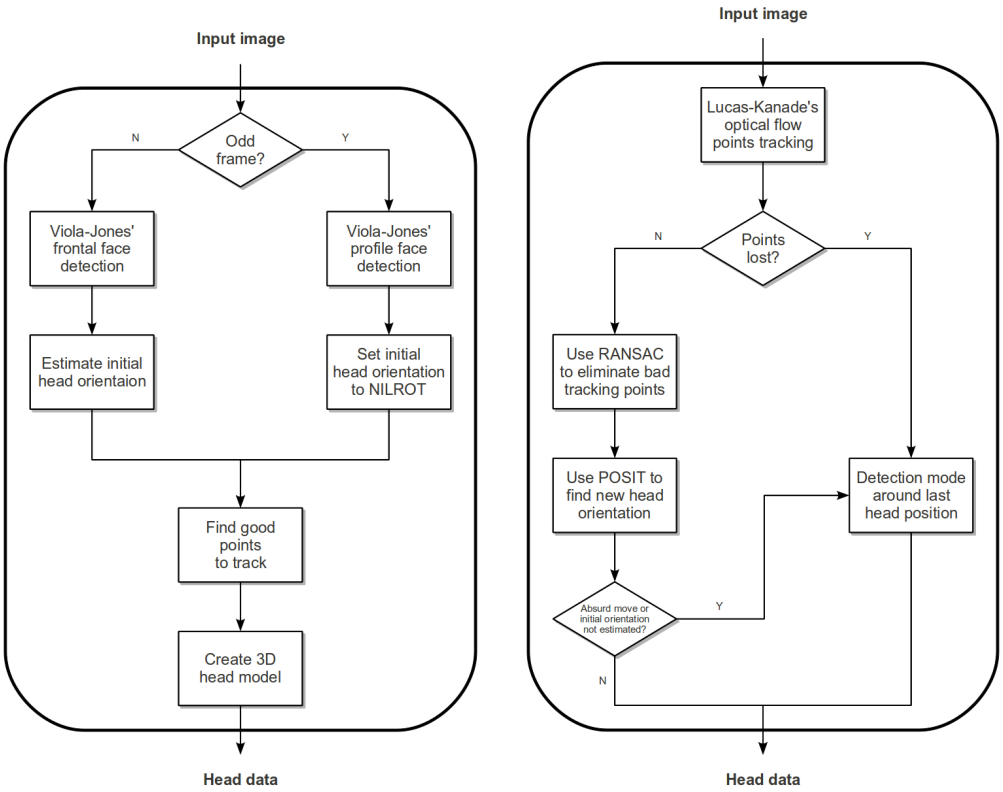
Detection mode: La fase di individuazione si ha all'inizio dell'esecuzione e ogni qualvolta venga persa una faccia. Fa uso principalmente di due classificatori Viola-Jones (3.1) per l'individuazione di visi frontali o di profilo. Il loro utilizzo avviene ad istanti alternati per non appesantire eccessivamente la computazione.

Sempre per motivi computazionali, il detector viene utilizzato in modo da restituire solo il viso più grande eventualmente presente nella scena, mentre visi più piccoli di 80x80 pixel vengono scartati a priori perchè troppo poveri di informazioni per effettuare un tracking accettabile.

Individuato un viso si cerca con metodi euristici di stabilire l'orientamento iniziale della testa (3.2), se ciò non è possibile si prosegue nell'esecuzione tenendo presente che il modello 3D che si andrà a creare non è una rappresentazione realistica di un viso frontale.

A questo punto si cercano all'interno dell'area individuata dei punti che saranno utili per il tracking successivo e, a partire dalla posizione di questi, li si

associa ad un modello cilindrico che approssimi la forma di una testa indipendentemente dall'orientamento iniziale. Si tratta di una approssimazione molto forte, che tuttavia nei nostri esperimenti è risultata accettabile per l'uso che ne si fa all'interno del tracker.



(a) *Detection mode.* (b) *Tracking mode.*

Figura 3.2: Le due modalità

Tracking mode: Una volta individuato un insieme di punti appartenenti ad una faccia, di cui si abbia anche un modello 3D, entra in gioco la fase di inseguimento. Per prima cosa, attraverso un algoritmo di Optical Flow (3.3) secondo l'implentazione a piramide di Lucas-Kanade (3.3.1), si individua la posizione dei punti da tracciare nel nuovo frame. Se il numero matching trovati é ritenuto insufficiente si tenta di ricampionare i punti applicando l'algoritmo di detection visto in precedenza, ma solo in un intorno dell'ultima posizione valida del viso.

Altrimenti, si cerca di filtrare i punti trovati attraverso un algoritmo euristico, il RANSAC (3.4), per la ricerca di omografie ottime, il quale non solo elimina i punti che non sono conformi alla trasformazione trovata, ma restuisce anche il nuovo orientamento del viso.

Infine, per migliorare la robustezza, si effettua un resampling dei punti nel caso il nuovo orientamento non sia compatibile con le normali pose facciali o sia avvenuta una rotazione troppo veloce, o semplicemente sia trascorso molto tempo dal'ultimo campionamento.

3.1 Detector Viola-Jones

I recenti face detector di tipo implicito hanno ridotto drammaticamente i tempi di computazione nonostante gli alti livelli di accuratezza. Viola-Jones, in particolare, si basa sull'idea di una boosted cascade di classificatori deboli.

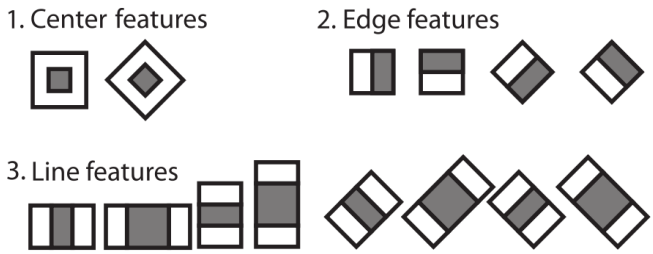


Figura 3.3: Classificatori deboli

In una fase offline di apprendimento, viene allenato un nuovo classificatore applicando in serie alcuni classificatori deboli ad un set sufficientemente grande di immagini contenenti l'oggetto da cercare (congiuntamente ad alcuni falsi positivi), in pose e con illuminazioni differenti per renderlo il più possibile indipendente da questi fattori. In questo modo, ad ogni iterazione, il classificatore è in grado di scartare una certa percentuale di pattern che non rappresentano gli oggetti cercati (e che erano stati accettati negli stadi precedenti). Come esempio, le feature riconosciute per il primo passo di un face-detector frontale, sono:

Figura 3.4: Classificatore frontale

La frequenza di rilevamento risultante, che chiameremo D, e la frequenza di falsi positivi, F, sono ottenute dalla combinazione delle performance di ogni singolo classificatore nella cascata:

$$D = \prod_{i=l}^{K} d_i \qquad F = \prod_{i=l}^{K} f_i \tag{3.1}$$

Seguendo questo approccio, e dato un rilevatore a 20 passi progettato per scartare ad ogni passo il 50% dei pattern che non identificano oggetti(frequenza di falsi positivi) ed eliminare solamente lo 0.1% dei pattern che al contrario individuano oggetti (frequenza di rilevamenti positivi), la frequenza stimata di rilevamento è ~ 0.98, con una quantità di falsi positivi quantificabile in 0.9×10^{-6}.

Una volta ottenuto il clasificatore, lo si applica all'immagine desiderata, nel nostro caso un frame della telecamera, e l'algoritmo cerca esaustivamente per ogni posizione e ad ogni possibile risoluzione una regione d'immagine che corrisponda al pattern del classificatore.

Sia per il training che per l'esecuzione, si utilizzano rappresentazioni integrali dell'immagine che permettono di calcolare e confrontare più velocemente somme di pixel comprese fra aree rettangolari.

$$sum(X, Y) = \sum_{x \leq X} \sum_{y \leq Y} image(x, y) \tag{3.2}$$

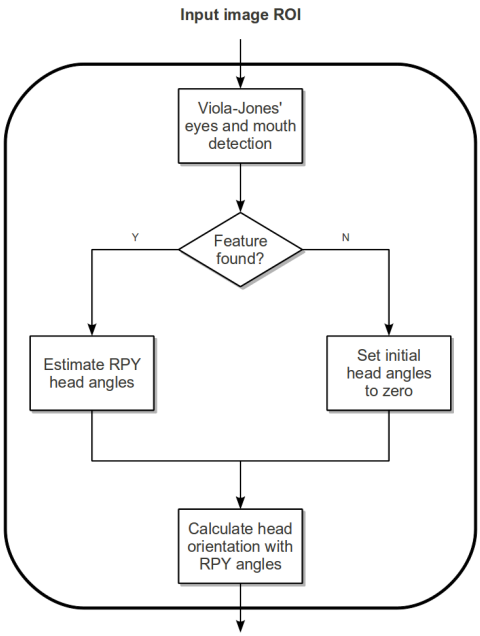
Questo schema permette di ottenere una velocità di computazione elevata, dovuta al fatto che le regioni sullo sfondo dell'immagine sono rapidamente scartate, e viene speso più tempo per le regioni dove è più probabile trovare gli oggetti cercati. Dunque, il designer del rilevatore deve solamente scegliere il numero di passi da far compiere al detector, assieme alle frequenze di falsi postivi e rilevamenti corretti, cercando un equilibrio tra accuratezza e velocità di esecuzione.

3.2 Stima orientamento

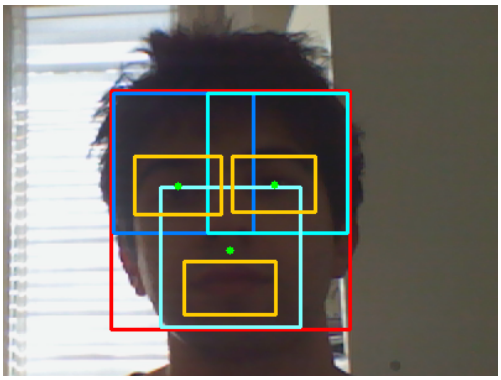
La stima dell'orientamento si basa sul rilevamento di feature utili all'interno del viso trovato, in particolare, si fa uso della posizione degli occhi e della punta del labbro superiore. Per l'individuazione si utilizza anche in questo caso l'algoritmo di Viola-Jones con classificatori specializzati nel rilevamento dell'occhio destro e sinistro e della bocca.

Si fa poi un rifinimento della posizione della pupilla all'interno dell'occhio attraverso tecniche di sogliature del colore (la pupilla è la zona più scura dell'occhio) e di calcolo del centro di massa del bianco.

Infine, si estraggono gli angoli di Roll, Pitch e Yaw della testa attraverso semplici considerazioni geometriche.



(a) *Diagramma funzionamento.*



(b) *In rosso la faccia rilevata, nelle tre tonalità di blu le zone di ricerca delle features, in giallo le features trovate, in verde le pupille e il labbro superiore.*

(c) *In ordine da sinistra a destra: l'occhio, l'occhio post-equalizzazione, l'occhio binarizzato, il centro di massa.*

Figura 3.5: Immagini relative alla stima dell'orientamento

3.3 Optical Flow

La capacità di tracking di un oggetto generico si riconduce spesso all'individuazione del movimento tra due frame successivi catturati dalla stessa sorgente, senza alcun dato sul contenuto di detti frame. Tipicamente, è il moto stesso ad indicare che sta accadendo qualcosa nella scena in esame. Il meccanismo dell'Optical Flow (termine che riassume il concetto precedente) è illustrato in figura.

E' possibile associare una velocità ad ogni pixel nell'immagine o, equivalentemente, un valore che rappresenti la distanza percorsa da un pixel tra il frame precedente e quello corrente. Questa modellizzazione è spesso definita come *optical flow denso*. All'atto pratico, un concetto apparentemente banale come quello dell'optical flow risulta però di difficile implementazione. Consideriamo il moto di un foglio di carta bianco. Molti dei pixel bianchi in un frame rimarrano probabilmente tali anche in quello successivo. Saranno solamente i margini del foglio a cambiare, e solo quelli perpendicolari alla direzione del moto. Il risultato è che i metodi densi debbano prevedere dei metodi di interpolazione tra i punti

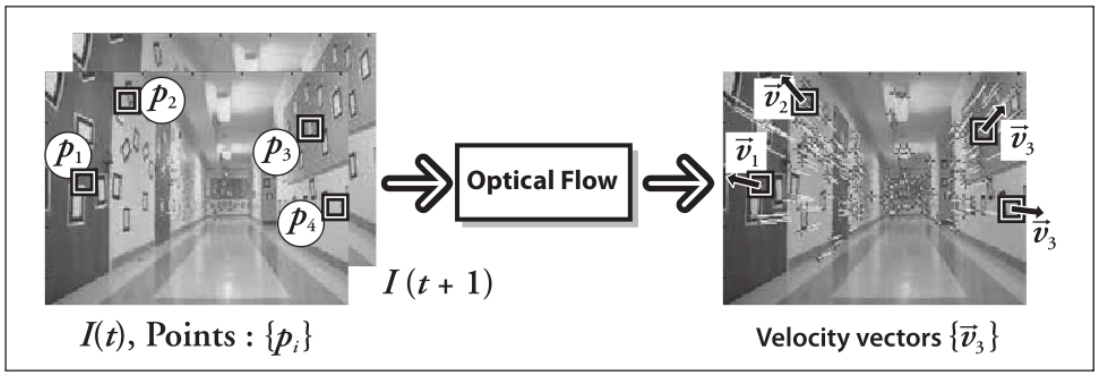


Figura 3.6: Esempio di funzionamento di optical flow denso

tracciabili più facilmente, così da rendere più semplice l'individuazione dei punti con caratteristiche più ambigue.

Questo ci porta a valutare un'altra opzione, l'*optical flow sparso*. Gli algoritmi di questa natura si fondano sulla capacità di specificare in anticipo i punti da tracciare. Se questi punti soddisfano determinate proprietà (ad esempio punti d'angolo) allora il tracking risulterà più agevole, robusto e preciso. Per molte applicazioni pratiche, il costo computazionale del tracking sparso è talmente inferiore a quello denso da relegare quest'ultimo ad ambiti squisitamente accademici.

3.3.1 Algoritmo Lucas-Kanade

L'algoritmo Lucas-Kanade (LK), originariamente proposto come metodo per ottenere risultati densi, è divenuto, per la sua facilità di applicazione, un'importante tecnica di analisi sparsa. Tale successo discende dal fatto che l'algoritmo si basa unicamente sull'informazione locale, derivata da una finesta che racchiude ogni singolo punto di interesse.

Lo svantaggio di utilizzare piccole finestre locali è che movimenti eccessivi della camera che acquisisce l'immagine possono portare i punti individuati dall'algoritmo al di fuori dei margini della finestra di individuazione, rendendoli impossibili da ritrovare.

Questo problema ha portato, nel corso degli anni, allo sviluppo di un algoritmo LK di natura "piramdiale", che effettua il tracking partendo dal livello più alto (minor dettaglio) di una piramide di immagini, per proseguire poi verso i livelli inferiori (dettaglio più elevato). Il Tracking effettuato utilizzando questo

accorgimento permette di evitare la perdita di punti dovuta a movimenti ampi e improvvisi.

3.3.2 Funzionamento dell'algoritmo

L'idea di fondo dell'algoritmo si basa su tre assunzioni:

1. Luminosità costante. Un pixel riferito ad un oggetto nell'immagine non cambia aspetto nel movimento (eventuale) tra frame successivi. Per immagini in scala di grigio (LK può essere effettuato anche a colori) ciò significa assumere che la luminosità di un pixel non cambi da frame a frame.

$$f(x,t) \equiv I(x(t),t) = I(x(t+dt),t+dt) \tag{3.3}$$

$$\frac{\partial f(x)}{\partial t} = 0 \tag{3.4}$$

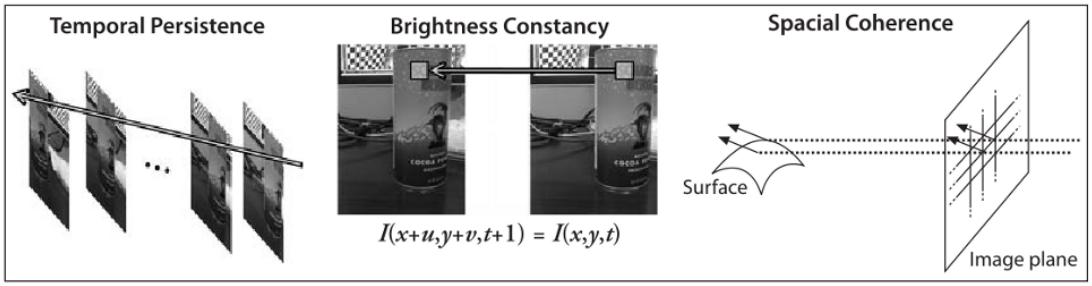
2. Persistenza temporale o piccoli movimenti. Il movimento di una porzione di immagine tra frame successivi è sufficientemente lento, o, meglio, gli incrementi temporali sono sufficientemente rapidi da poter supporre che il movimento di un oggetto sia piccolo da frame a frame.

3. Coerenza spaziale. In un'immagine, i punti vicini appartengono alla stessa superficie, hanno movimenti simili, ed hanno in generale un comportamento "rigido".

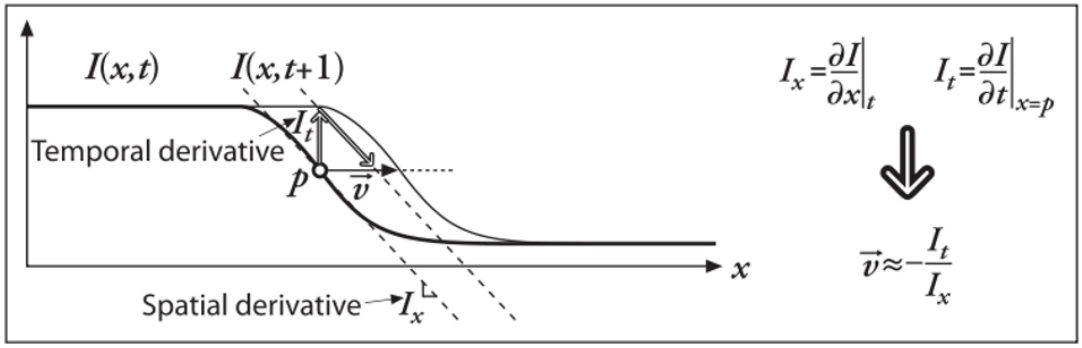
Analizziamo ora l'impatto di queste assunzioni relativamente al caso monodimensionale. Possiamo partire con l'equazione di consistenza della luminosità, e sostituire la definizione di luminosità *f(x,t)*, tenendo in considerazione la dipendenza implicita di x da t,$I(x(t),t)$, ed infine applicare la regola a catena per la derivazione parziale. Questo porta a

$$\underbrace{\left.\frac{\partial I}{\partial x}\right|_t}_{I_x} \underbrace{\left(\frac{\partial I}{\partial x}\right)}_{v} + \underbrace{\left.\frac{\partial I}{\partial t}\right|_{x(t)}}_{I_t} = 0 \tag{3.5}$$

dove I_x è la derivata spaziale per la prima immagine, I_t è la derivata tra le immagini al variare del tempo, e v è la velocità che stiamo cercando. Arriviamo dunque alla semplice equazione per l'optical flow nel caso monodimensionale:



(a) *Assunzioni.*



(b) *Optical flow in una dimensione.*

Figura 3.7

$$\mathbf{v} = -\frac{I_t}{I_x} \tag{3.6}$$

Generalizzando il procedimento visto alle immagini bidimensionali, l'equazione da risolvere diventa:

$$I_x u + I_y v + I_t = 0 \tag{3.7}$$

Sfortunatamente, l'equazione ha due incognite per ogni pixel. Il problema è dunque risolvibile per la componente di moto normale alla linea descritta dall'equazione.

L'optical flow 'normale' è strettamente legato al problema dell'apertura, che si presenta quando si ha una finesta di dimensioni ridotte entro cui operare l'individuazione del moto. Quando il movimento è rilevato in un finestra ridotta, nella maggior parte dei casi è visibile solamente un bordo, e non un angolo. Ma un singolo bordo non è sufficiente per determinare esattamente in quale direzione l'oggetto si stia effettivamente muovendo. Viene in questo caso in aiuto l'ultima assunzione effettuata, ossia che se un gruppo di pixel collegati si muove in manie-

ra rigida, allora il movimento del pixel centrale può essere ottenuto utilizzando i pixel circostanti per creare un sistema di equazioni, come descritto dalla formula seguente (nel caso di finestra 5x5).

$$\underbrace{\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix}}_{A=25\times 2} \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{d=2\times 1} = -\underbrace{\begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}}_{b=2\times 1} \tag{3.8}$$

Siamo allora arrivati ad avere un sistema con un numero di vincoli sufficienti a permettere la risoluzione (posto che nella finestra ci sia un numero di bordi maggiore di 1). Per risolvere il sistema, si riduce la formula allo scarto quadratico medio, dove $\min\|Ad - b\|^2$ è risolto come:

$$\underbrace{(A^{\mathrm{T}}A)}_{2\times 2}\underbrace{d}_{2\times 1} = \underbrace{A^{\mathrm{T}}b}_{2\times 2} \tag{3.9}$$

Da questa relazione si ottengono le componenti del moto u e v. Nel dettaglio:

$$\underbrace{\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}}_{A^{\mathrm{T}}A} \begin{bmatrix} u \\ v \end{bmatrix} = -\underbrace{\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}}_{A^{\mathrm{T}}b} \tag{3.10}$$

La soluzione dell'equazione è allora:

$$\begin{bmatrix} u \\ v \end{bmatrix} = (A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}b \tag{3.11}$$

In quali casi l'equazione è risolvibile solamente quando $(A^{\mathrm{T}}A)$ è invertibile ed $(A^{\mathrm{T}}A)$ è invertibile quando ha rango massimo (s), ossia quando ha due autovettori i cui valori numerici sono grandi. Questo accadra in quelle regioni dell'immagine che includano texture in movimento in almeno due direzioni.

Dai risultati esposti, si evince che lo svantaggio principale dell'algoritmo è nei movimenti repentini e ampi, abbastanza comuni per le telecamere operanti attorno ai 30Hz. Anche supponendo di utilizzare finestre ampie per individuare i movimenti più grandi, verrebbe a mancare la fondamentale assunzione sul moto (che si ricorda debba essere il più possibile regolare) fatta all'inizio della trattazione. Per eliminare il problema, è pratica comune utilizzare una tecnica a piramide, dove si comincia ad analizzare il moto su grande scala (con grandi

finestre) e si raffinano poi le assunzioni iniziali procedendo nella piramide fino al livello dei pixel.

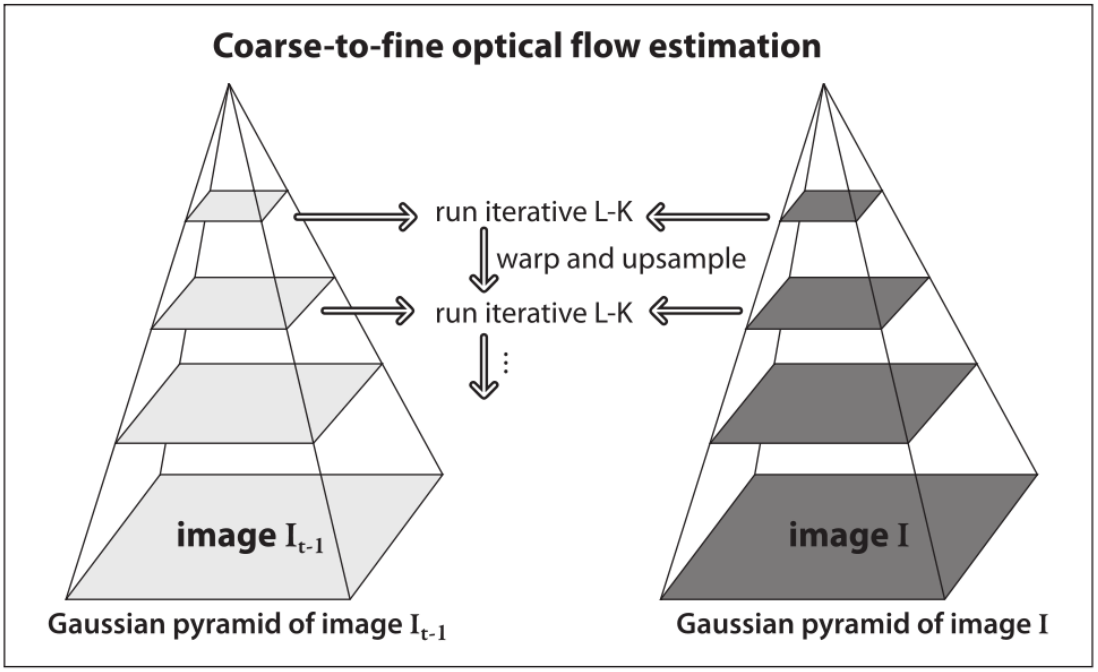


Figura 3.8: Optical flow piramidale

3.4 Algoritmo RANSAC per omografia

Verrà ora trattato il problema del riconoscimento dell'omografia tra due immagini utilizzando tecniche esplicite, in particolare tra frame successivi catturati da una medesima sorgente. Nello specifico, si utilizzerà l'algoritmo RANSAC (Random Sample Consensus), che risulta essere il più robusto tra quelli attualmente a disposizione.

Il processo computazionale consta di due passi fondamentali: il primo consiste nell'ottenere l'omografia ottima tra i punti corrispondenti nelle due immagini, mentre il secondo si occupa di valutare l'effettiva consistenza delle corrispondenze e, conseguentemente, eliminare quei punti che non rispettano l'omografia ottima trovata.

3.4.1 Struttura dell'algoritmo RANSAC

Date in ingresso n presunte corrispondenze tra due immagini.

- inizializza il numero di stime N e le soglie T_DIST, $MAX_inlier = -1$, $MIN_std = 10e5$ e $p = 0.99$
- per la i-esima stima ($i = 1 : N$)
  - **(a)** scegli casualmente 4 corrispondenze
  - **(b)** controlla se i punti sono allineati, se si, ritorna al passo precedente
  - **(c)** calcola l'omografia H_{curr} a partire dalle 4 coppie di punti tramite POSIT
  - **(d)** per ogni corrispondenza stimata, calcola la distanza d_i utilizzando H_{curr} appena calcolato
  - **(e)** calcola la deviazione standard della distanza tra punti allineati curr_std
  - **(f)** conteggia il numero di punti allineati m con distanza di $d_i < T\_DIST$
  - **(g)** se $m > MAX\_inlier$ aggiorna il miglior $H = H_{curr}$ e memorizza tutti i punti allineati
  - **(h)** aggiorna N. Calcola $\epsilon = 1 - m/n$ e setta $N = \frac{log(1-p)}{log(1-(1-4)^4)}$
- raffinamento: esegue nuovamente la stima di H con il set di migliori punti trovati al punto precedente, sempre tramite POSIT
- elimina tutti i punti aventi distanza $d > T\_DIST$ con l'H calcolata con il raffinamento

3.4.2 POSIT: stima di posizione e orientamento 3D

Illustriamo ora un utile algoritmo preposto alla stima della posizione di un oggetto noto nelle 3 dimensioni spaziali e del suo orientamento. POSIT (ossia Pose from Ortography and Scaling with Iteration) è un algoritmo proposto originariamente nel 1992 per calcolare la posizione (la posizione T e l'orientamento R descitti da 6 parametri) di un oggetto 3D di cui si conoscano le dimensioni esatte.

Per calcolare la posizione, è necessario individuare, nell'immagine, le posizioni corrispondenti ad almeno 4 punti non complanari sulla superficie dell'oggetto. La prima parte dell'algoritmo, *pose from orthography and scaling* (POS), assume che i punti sull'oggetto esaminato siano tutti alla stessa profondità (stessa distanza dalla sorgente di acquisizione dell'immagine) e che le variazioni di dimensione dal modello originale siano dovute unicamente ad una eventuale differenza di distanza dalla videocamera. In questo caso esiste una soluzione in forma chiusa del problema dell'orientamento basata sulla scalatura. L'assunzione che i punti siano alla

stessa profondità discende dal considerare l'oggetto ad una distanza sufficientemente grande da rendere trascurabile le differenze di profondità nell'oggetto stesso (questa assunzione è nota come approssimazione *weak perspective*).

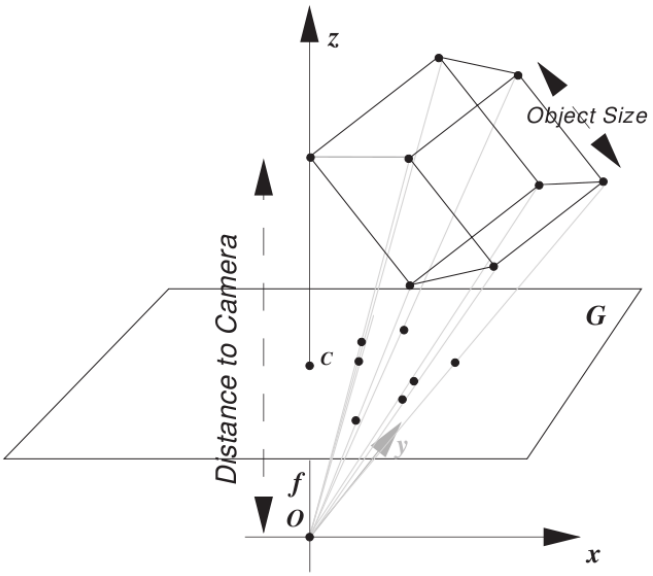


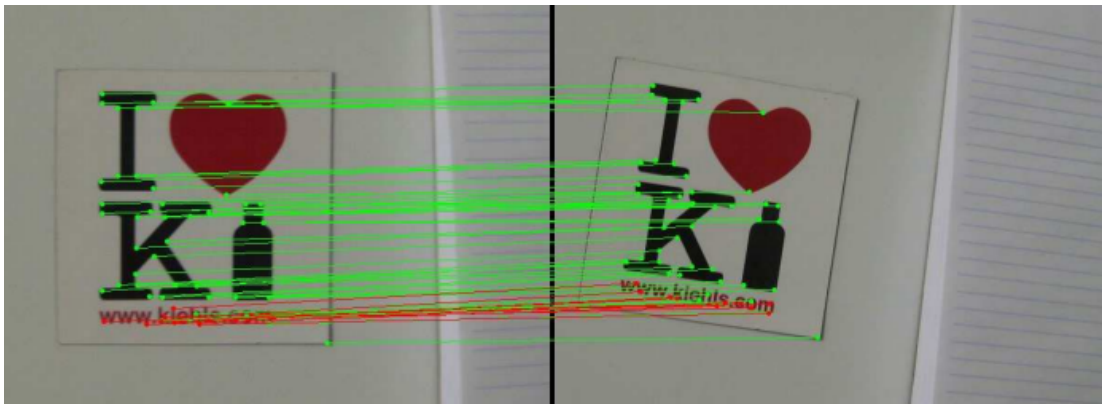
Figura 3.9: Assunzioni ed esempio di POSIT

Supponendo di conoscere le proprietà ottiche dell'obiettivo della telecamera, possiamo ricavare la scalatura prospettica del nostro oggetto e approssimarne dunque la posa. Il risultato non sarà molto accurato, ma possiamo stimare dove finirebbero i 4 punti considerati nel caso l'oggetto 3D fosse nella posa calcolata tramite algoritmo POS. Possiamo allora reiterare il processo con le posizioni di questi nuovi punti come input dell'algoritmo POS. Tipicamente, il procedimento converge alla posa reale entro quattro o cinque iterazioni (da cui il nome algoritmo POS con IT*erazione*).

Si ricordi, comunque, che il procedimento viene effettuato supponendo che le distanze di profondità interne all'oggetto siano piccole rispetto alla distanza dalla telecamera. Se questa assunzione non risultazze verificata, l'algoritmo non convergerebbe o convergerebbe in maniera errata. Esitono implementazioni di questo algoritmo che permettono di tracciare più di quattro punti non complanari, la loro accuratezza è dunque maggiore rispetto ad un algoritmo POSIT di base e risulta comoda (anche se non sempre precisa) per il raffinamento dell'orientamento della testa della persona inquadrata.

3.4.3 Risultati del RANSAC

In questa sezione verranno mostrati i risultati ottenuti applicando questo algoritmo. Mostreremo innanzitutto le immagini con le corrispendenze tra i punti, le linee verdi rappresentano i punti che rispettano l'omografia, mentre le linee rosse indicano i punti che non lo fanno. Si nota come l'algoritmo RANSAC elimini efficientemente le corrispondenze non accurate. Le immagini finali mostrano la prima immagine originale e la seconda trasformata utilizzando il parametro H ottenuto. Nell'esempio, lo scarto quadratico medio (MSE) tra al prima immagine originale e quella trasformata risulta essere di 85.71, più che accettabile per l'applicazione in cui la si utilizza.



(a) *Eliminazione false corrispondenze.*

(b) *Omografia applicata.*

Figura 3.10: Esempio applicazione RANSAC a due immagini sequenziali

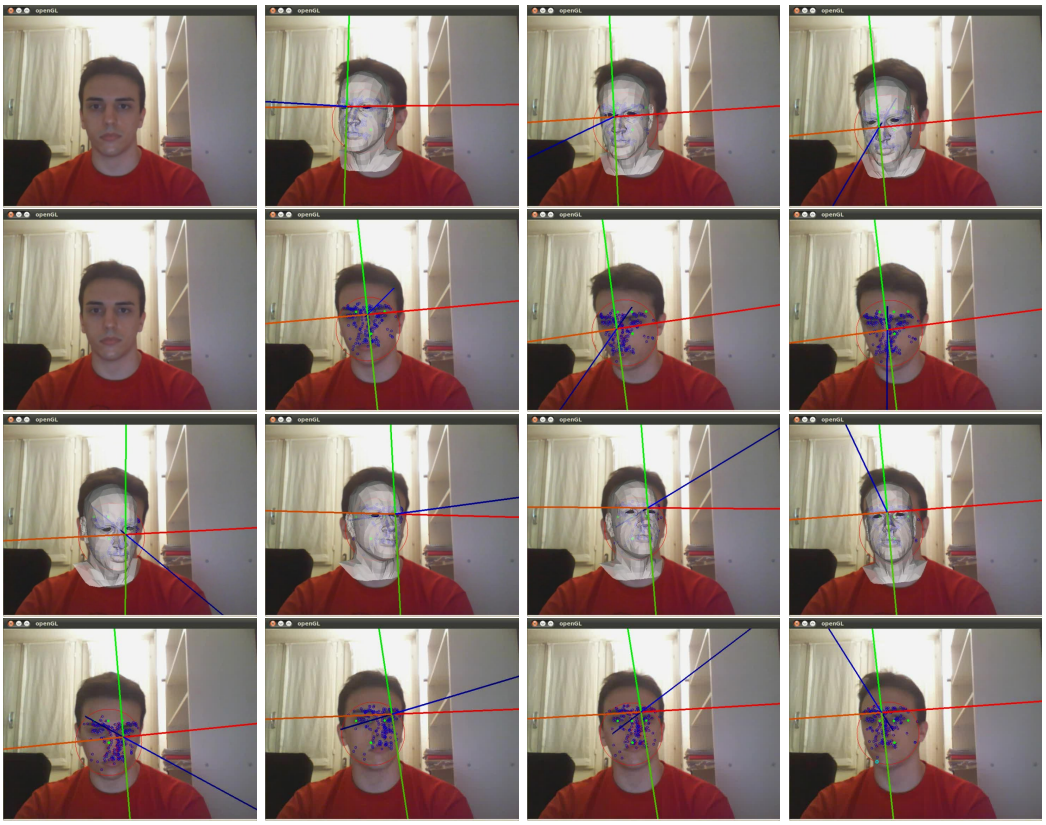


Figura 3.11: Sequenza di esecuzione dell'algoritmo con movimento circolare della testa e con rendering del modello 3D (righe 1 e 3) e senza rendering (righe 2 e 4)

Capitolo 4

L'Implementazione (Parte I)

L'algoritmo presentato si avvale, come è ovvio, di molte funzioni per l'analisi di immagini; una soluzione ottima potrebbe auindi essere quella di utilizzare per l'implementazione il linguaggio Matlab. Matlab è sicuramente uno dei migliori software per il calcolo matriciale (e di conseguenza l'analisi d'immagini), ma lo sviluppo di applicazioni real-time con questo linguaggio risulta impensabile. Per questo motivo, seguendo quanto già fatto in precedenza con E-2? e in approccio ingegneristicamente rilevante, abbiamo deciso ti utilizzare delle librerie libere scritte in C e C++: le OpenCV.

4.1 OpenCV

OpenCV è una libreria open source di compter vision. La libreria è scritta in C e C++ ed è possibile sviluppare applicazioni sotto qualsiasi sistema operativo, aspetto per noi fondamentale essendo il sistema funzionante totalmente in Linux.

La libreria è nata da un'iniziativa di ricerca di Intel per migliore applicazione fortemente stressanti per la CPU. Con questo obiettivo, vennero avviati diversi progetti, dal tracciamento di laser alla visualizzazione di immagini 3D. Durante una visitÃ in un' università, uno degli sviluppatori decise di applicare la stessa politica di passaggio libero del codice tra persone per migliorare delle librerie già esistenti di computer vision, evitando di dover riscrivere il codice ex novo. Iniziò così un progetto finalizzato all'ottenimento di un' infrastruttuta universale ed efficiente per la computer vision.

Proprio per la natura iniziale del progetto di Intel, OpenCV è stata pensata con un occhio di riguardo alla complessità computazionale e un utilizzo in applicazioni real-time. La libreria è ottimizzata per C e può avvalersi dei vantaggi di

processori multi-core. Essa contiene oltre 500 funzioni che si occupano di varie aree dell'analisi d'immagini, tra cui la calibrazione delle video vamera, la visione stereoscopica, l'estrazione di informazioni utili.

4.2 Funzioni

L'algoritmo è organizzato all'interno di una classe, FaceDetector, in modo da rendere l'implementazione trasparente al programma (nel nostro caso il VisionFrontalExpert) che la utilizza. L'interfaccia pubblica della classe è la seguente:

```
1  public:
2    FaceDetector();
3    FaceDetector( String frontalCascadeName,
4                  String profileCascadeName,
5                  String rEyeCascadeName,
6                  String lEyeCascadeName,
7                  String mouthCascadeName );
8
9    Point getPoint();
10   int getRadius();
11   void getGLOrientation( float& r, float& p, float& y );
12   void getGlPositMatrix( double myGlPositMatrix[16] );
13
14   bool faceDetect( unsigned char* img_pointer,
15                    double scale, double angle );
```

Oltre ai costruttori della classe, sono presenti dei getter che possono essere usati per recuperare, rispettivamente, i riferimenti in pixel del centro e del raggio del cerchio che meglio approssima il viso, nonchè l'orientamento della testa nello spazio, sia come angoli di Roll-Pitch-Yaw che come matrice di Roto-Traslazione.

La funzione principale è però faceDetect, che andrebbe chiamata ad ogni acquisizione di un nuovo frame. Una volta in esecuzione, entrano in gioco automaticamente le due modalità di funzionamento (ricerca o tracking) in base alle informazioni dei frame precedenti. La funzione restituisce vero se è stata rilevata (o tracciata) una faccia, falso altrimenti. In base al valore restituito, l'esperto deciderà di utilizzare o meno i getter per avere le informazioni necessarie sull'obiettivo.

Il funzionamento in modalità di rilevamento o di tracciamento è contenuto in funzioni private della classe non accesibili dall'utilizzatore esterno. La loro

implmentazione non è troppo complessa e gl'unici parametri che potrebbe essere utile modificare sono:

- il numero minimo di punti (*MIN_POINTS*) sotto il quale si ritiene il volto perso
- il numero minimo di punti (*MIN_POINTS_POSIT*) per cui si ritiene possibile applicare l'algoritmo RANSAC (deve essere magiore di *RANSAC_SAMPLES*)
- la percentuale di punti in movimento (*POINT_PERC*) che deve essere superata affinchè si possa dire che la faccia si sia spostata
- gl'angoli massimi di pitch e di roll (*MAX_ANGLE*) che rendono una faccia non plausibile (l'angolo di yaw, invece, va potenzialmente da 0 a 360 gradi)
- il numero di iterazioni (*OLD_FACE* e *MAX_FACE*) oltre le quali un volto è ritenuto non più attendibile e necessita di un ricampionamento
- il numero di iterazioni (*MAX_LOST*) oltre le quali un volto che sia uscito dal campo visivo è ritenuto perso

Inoltre, è in queste parti del codice che entrano in gioco gli algoritmi descritti in precedenza, in particolare, l'"Haar Detection" nel rilevamento mentre l'"Optical Flow", il "RANSAC" e il "POSIT" nel tracciamento. Per permetterne un riutilizzo in altri ambiti queste sono definite come funzioni statiche utilizzabili senza creare un'istanza della classe. Diventano quindi importanti i parametri passati a queste funzioni presentate a seguire.

4.2.1 Haar detection

L'algoritmo di Viola-Jones viene utilizzato anche nell'indivduazione delle features all'interno del volto, oltre che nella fase di rilevazione e in quella di correzione del volto stesso. Esiste una funzione apposita in OpenCV per questo algoritmo (*detectMultiScale*) che abbiamo deciso di wrappare nel modo seguente:

```
1 Rect detectBiggestHaar(Mat img, CascadeClassifier cascade,
2                        Size cascadeSize, int newWidth)
```

La funzione restituisce la porzione d'immagine (come rettangolo) dell'oggetto più grande cercato, se non trovato restituisce un rettangolo di larghezza 0. Il significato dei parametri è abbastanza intuitivo:

- *img* è l'immagine in cui cercare (in bianco e nero, non necessariamente intera)
- *cascade* e cascadeSize sono i dati sul classificatore da utilizzare (vedi seguito)
- *newWidth* è la grandezza a cui si vuole riscalare l'immagine, utile per cercare in immagini più piccole della risoluzione massima e per fare una scalatura uniforme nella ricerca delle features interne

4.2.2 Classificatori

Nell'algoritmo vengono utilizzati 5 classificatori diversi: forntale, di profilo, occhio destro, occhio sinistro e bocca. Il classificatore di profilo essendo simmetrico può essere utilizzato (avendo l'accortezza di ribaltare l'immagine lungo l'asse y) sia per il profilo destro che per il sinistro.

OpenCV, nell'ultima versione disponibile, offre già questi classificatori e per alcuni di essi ne esistono più versioni. La nostra scelta si è, quindi, basata sull'analisi effettuata in CIT in cui si suggerisce di usare i seguenti classificatori:

- *haarcascade_frontalface_alt2.xml* 40x40
- *haarcascade_profileface.xml* 40x40
- *haarcascade_mcs_righteye.xml* 18x12
- *haarcascade_mcs_lefteye.xml* 18x12
- *haarcascade_mcs_mouth.xml* 18x15

I valori accanto ai classificatori sono la larghezza della finestra con cui sono stati allenati. Nel codice si utiliazzano sempre i valori corrispondenti a ciascun classificatore, infatti, usarne di minori sarebbe inutile perchè si perderebbero molte informazioni acquisite con l'allenamento fatto. D'altro canto anzichè usarne di maggiori è preferibile rimpicciolire l'immagine così da dover gestire un numero inferiore di pixel.

4.2.3 Optical Flow

Anche per l'algoritmo di Oprical Flow, nella versione a piramide di Lucas-Kanade, è già implementato in OpenCV (*cvCalcOpticalFlowPyrLK*) e, anche in questo caso si è deciso di wrapparla perchè utilizza ancora delle strutture dati di versioni precedenti di OpenCV poco efficienti in C++:

```
void trackOptFlow( Mat prev_gray, Mat gray,
                   IplImage* prev_pyramid,
                   IplImage* pyramid,
                   vector<Point2f>& points,
                   vector<Point2f>& prev_points,
                   int& flags, int& changed,
                   int& notStatus, float& totErr );
```

- *prev_** sono le variabili riferite al frame precedente
- *gray* è il frame convertito in bianco e nero
- *pyramid* è il buffer per l'analisi piramidale
- *points* è il vettore di punti di interesse
- *flags* indica se la piramide precedente è stata o meno precalcolata (deve valere 0 alla prima iterazione e 1 nelle successive)
- *changed* conterrÃ il numero di punti la cui distanza dalla posizione precedente sia maggiore di una certa soglia
- notStatus conterrÃ il numero di punti che si Ã¨ stati in grado di tracciare
- *totErr* conterrÃ l'errore medio che l'algoritmo compie nel tracciamento

4.2.4 RANSAC

L'algoritmo RANSAC a differenza degli altri e del POSIT (che viene utilizzato all'interno dello stesso) non Ã¨ implentato in OpenCV. Si sono quindi utilizzate le indicazioni presenti in CIT e in CIT. Il risultato, al di lÃ delle implementeazioni di supporto interne, Ã¨ il seguente:

```
int ransac( vector<Point2f> imagePoints,
            vector<Point3f> objectPoints,
            CvMatr32f rotation_matrix,
            CvVect32f translation_vector,
            vector<int>& outliers )
```

- *imagePoints* è il vettore di punti riferiti al centro del frame

- *objectPoints* è il vettore dei punti corrispondenti del modello 3D
- *rotation_matrix* e *translation_vector* conterrano l'omografia che meglio descrive lo spostamento dei punti rispetto al campionamento iniziale
- *outliers* conterrÃ la posizione dei punti d'interesse all'interno del vettore che li contiene che non rispettano l'omografia e andranno rimossi

In questo caso, inoltre, sono ridefinibili alcuni parametri fondamentali:

- il numero minimo di punti (*RANSAC_SAMPLES*) per cui l'algoritmo è applicabile, se non rispettato la funzione ritorna 1, altrimenti 0
- la soglia (*RANSAC_DISTANCE_THRESHOLD*) che la distanza dei punti proiettati con l'omografia rispetto ai punti del modello 3D deve rispettare affinchÃ¨ questi siano considerati validi
- il numero di iterazioni (*RANSAC_ITERATIONS*) per la ricerca della migliore omografia

Capitolo 5

La Logica Fuzzy

La logica Fuzzy (LF) fu ideata inizialmente da Lofti A. Zadeh, professore di scienze informatiche alla Univeristy of California di Berkley. Si tratta fondamentalmente di una logica multivalore, che permette di definire dei valori intermedi tra quelli convenzionali di vero e falso, si/no, alto/basso, ecc. Nozioni quali *piuttosto basso* o *molto veloce* possono essere così formulate matematicamente e processate da calcolatori, in modo da adottare, nella programmazione, un'approccio più simile a quello umano.

I sistemi fuzzy sono un'alternativa alle tradizionali nozioni di appartenenza insiemistica e logica che hanno avuto origine nella filosofia greca antica. L'accuratezza della matematica come scienza deve il suo successo agli sforzi di Aristotele e dei filosofi che lo hanno preceduto. Nel loro tentativo di delineare una concisa teoria della logica, e successivamente della matematica, furono postulate le cosiddette "Leggi del pensiero".

Una di queste, la regola del terzo escluso, afferma che ogni proposizione debba necessariamente essere vera o falsa. Anche quando Parmenide propose la prima versione di tale legge (attorno al 400 A.C) le proteste furono numerose e immediate: Eraclito, ad esempio, teorizzò la possibilità della coesistenza di verità e falsità riferite a un medesimo soggetto.

Fu tuttavia Platone a porre le fondamenta di quella che sarebbe diventata la logica fuzzy, indicando una terza regione (oltre al vero ed al falso) dove gli opposti "si combattevano". Altri, più recenti filosofi hanno fatto eco ai suo sentimenti, in particolare Hegel, Marx ed Engels. Ma è stato Lukasiewicz a proporre per primo un'alternativa sistematica alla logica bivalente aristotelica. La logica fuzzy è emersa come uno strumento utilissimo sia per il controllo e la modifica di sistemi e complessi processi industriali, sia per l'elettronica e l'intrattenimento domestico.

Anche applicazioni avanzate come la classificazione dei dati in immagini SAR hanno tratto vantaggio dalla sua applicazione.

5.1 Insiemi fuzzy e insiemi definiti

La nozione basilare di un sistema fuzzy è quella di (sotto)insieme fuzzy. Nella matematica classica siamo abituati a trattare con quelli che vengono chiamati insiemi definiti. Ad esempio, i valori g possibili di coerenza interferometrica sono l'insieme X di tutti i numeri reali tra 0 e 1. Da questo insieme X si può definire un sottoinsieme A, (ad esempio tutti i valori $0 \leq g \leq 0.2$). La funzione caratteristica di A, (ossia la funzione che assegna un 1 o uno 0 ad ogni elemento di X, a seconda che questo appartenga o no al sottoinsieme A) è mostrata nell'immagine seguente.

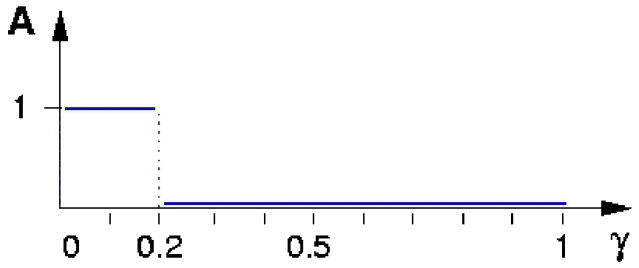


Figura 5.1: Funzione caratteristica di un insieme canonico

Gli elementi a cui è stato assegnato il numero 1 possono essere interpretati come gli appartenenti ad A, mentre gli elementi a cui è stato assegnato 0 sono quelli esclusi dall'insieme . Questo concetto è sufficiente per molti campi di applicazione, ma per altri ambiti è evidente la sua scarsa flessibilità.

Ad esempio, è noto che l'acqua mostri una bassa coerenza interferometrica g nelle immagini SAR. Siccome g parte da 0, il range inferiore di questo insieme deve essere vuoto. Quello superiore, d'altro canto, è piuttosto difficile da definire.

Come primo tentativo, impostiamo il range superiore a 0.2. Abbiamo dunque l'intervallo definito $B = [0, 0.2]$; ma ciò significa che un valore g di 0.20 è basso, mentre un valore di 0.21 non lo è. Ovviamente, questo è un problema strutturale, poiché spostando il limite superiore del range da $g = 0.20$ ad un punto arbitrario, la questione rimane irrisolta.

Un modo più naturale di costruire l'insieme B potrebbe consistere nello sfumare la linea di separazione tra alto e basso. Questo può essere fatto permettendo si l'esistenza delle (definite) decisioni SI e NO, ma aggiungendovi anche altre

regole quali *abbastanza bassso*. Un insieme fuzzy ci permette di definire tale nozione. L'obiettivo è quello di utilizzare insiemi fuzzy per rendere i computer più *intelligenti*, l'idea precedente va quindi formalizzata con maggiore attenzione.

Nell'esempio, a tutti gli elementi era assegnato il valore 0 o 1. Un modo semplice per generalizzare questo concetto è permettere più valori tra 0 e 1. In effetti, possono essere permesse infinite altre alternative tra i limiti superiore e inferiore, e precisamente tutti i valori dell'intervallo $I = [0, 1]$. L'interpretazione dei numeri, assegnati ora a tutti gli elementi, diventa molto più complessa. Naturalmente, il valore 1 assegnato ad un elemento, sta a significare che esseo appartiene all'insieme B, mentre 0 significa che non appartiene a B.

Tutti gli altri valori identificano un'appartenenza graduale a B. La funzione di appartenenza è una rappresentazione grafica della misura di partecipazione di ciascun input. Essa associa un peso ad ogni input processato, definisce le sovrapposizioni tra input, ed infine determina il valore di output. Le regole usano il valore di appartenenza dell'imput come fattore di peso per determinarne l'influenza nell'inieme fuzzy di output che rappresenta la conclusione finale.

La funzione di appartenenza, che opera in questo caso sull'insieme fuzzy di coerenza interferometrica g, restituisce un valore tra 0.0 e 1.0. Ad esempio, un valore di g di 0.3 ha un'appartenenza di 0.5 all'insieme di bassa coerenza. E' importante notare la differenza tra logica fuzzy e probabilità. Entrambe operano nello stesso range numerico, ed hanno valori simili: 0.0 rappresenta il falso (o la non apparteneza), e 1.0 è il vero (o appartenenza completa).

Tuttavia, c'è una grossa differenza tra le due affermazioni: l'approccio probabilistico porta all'affermazione in linguaggio naturale, "C'è il 50% di possibilità che g sia basso", mentre la terminologia fuzzy corrisponde a "il grado di appartenenza di g all'insieme di bassa coerenza interferometrica è 0.50". La differenza semantica è significativa: la prima prospettiva indica che g è o non è bassa, solamente non possiamo saperlo con certezza (ma solo con una probabilità del 50%). Al contrario, la terminologia fuzzy suppone che g sia *più o meno* bassa.

5.2 Operazioni sugli insiemi Fuzzy

Possiamo ora introdurre le operazioni di base sugli insiemi fuzzy. Similmente alle operazioni sugli insiemi definiti, potremmo voler intersecare, unire e negare insiemi fuzzy. Nel suo primo articolo sugli insiemi fuzzy, L. A. Zadeh suggerì l'operatore minimo per l'intersezione e l'operatore massimo per l'unione di due insiemi fuzzy. Si può mostrare come questi operatori coincidano con l'unione e

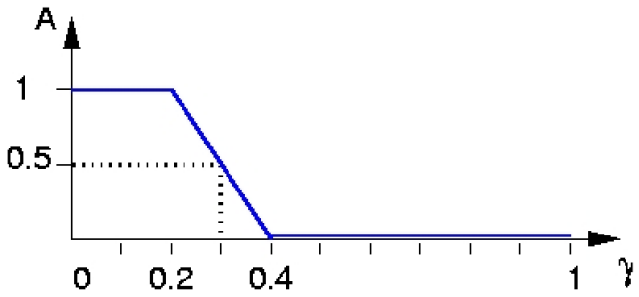


Figura 5.2: Funzione caratteristica di un insieme fuzzy

l'intersezione per insiemi definiti solo se consideriamo livelli di appartenenza di 0 e 1. Ad esempio, sia A è un intervallo fuzzy tra 5 e 8 e B un numero *circa 4* come mostrate in figura seguente

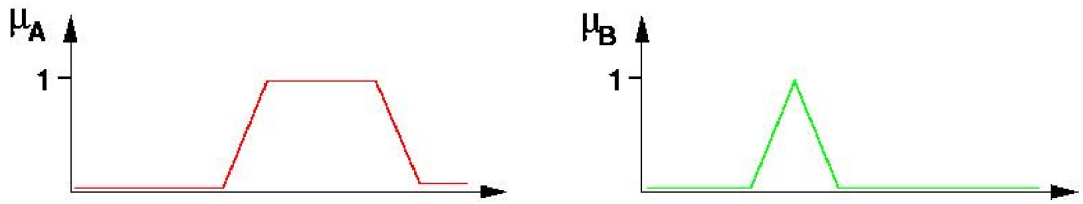


Figura 5.3: Esempi di inziemi fuzzy

In questo caso, l'insieme fuzzy *tra 5 E 8* e *circa 4* è

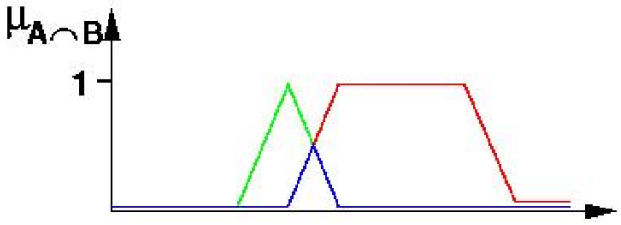


Figura 5.4: Operatore AND in logica fuzzy

In figura 5.5 sono mostrati gli insiemi *tra 5 E 8* O *circa 4* e l'insieme negazione di A

I classificatori "sfumati" sono una delle applicazioni della teoria Fuzzy. La conoscenza è usata e può essere espressa in modo molto naturale usando variabili linguistiche, che sono descritte da insiemi fuzzy. Ora. la conoscenza di queste variabili può essere formulata come regole del tipo

```
IF A basso AND B medio AND C medio AND D medio THEN Classe = classe4
```

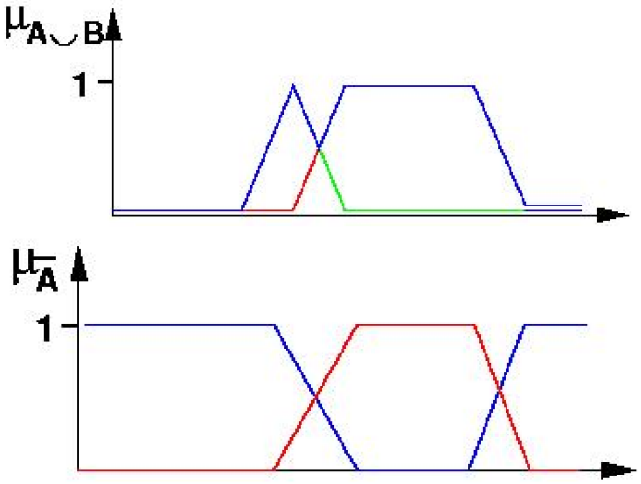


Figura 5.5: Operatore OR e negazione in logica fuzzy

Le regole possono essere combinate in una tabella di regole (*Tabella 5.1*)

Tabella 5.1: Esempio di regola fuzzy

R#	*feature A*	*feature B*	*feature C*	*feature D*	*classe*
1:	bassa	media	media	media	classe1
2:	media	alta	media	bassa	classe2
3:	bassa	alta	media	alta	classe3
4:	bassa	alta	media	alta	classe1
5:	media	media	media	media	classe4
...:	...	...	...	...	...
N:	bassa	alta	media	bassa	sconosciuta

Le regole linguistiche che descrivono il controllo del sistema consistono di due parti fondamentali: un blocco antecedente (tra IF e THEN) e uno conseguente (che segue THEN).

A seconda del sistema, potrebbe non essere necessario valutare ogni possibile combinazione di input, siccome alcune potrebbero verificarsi raramente o mai. Facendo questa semplice valutazione, è possibile valutare un numero inferiore di regole, semplificando il processo logico e migliorando le performance del sistema fuzzy.

Gli input sono combinati logicamente usando un operatore AND per produrre un output per tutti gli input attesi. Le conclusioni attive sono poi combinate

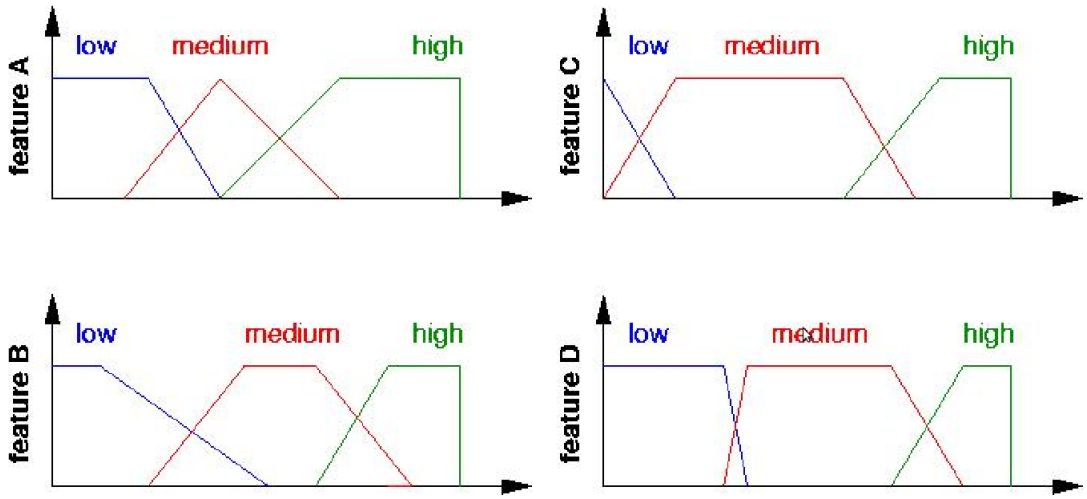


Figura 5.6: Esempio: variabili linguistiche

in una somma logica per ogni funzione di appartenenza. Tutto ciò che rimane da fare è combinare queste somme logiche in un processo di defuzzificazione per produrre un risultato definito. Ad esempio, per la coppia di ingresso $H = 35$ e $\alpha = 30$, verrebbe applicato lo schema seguente:

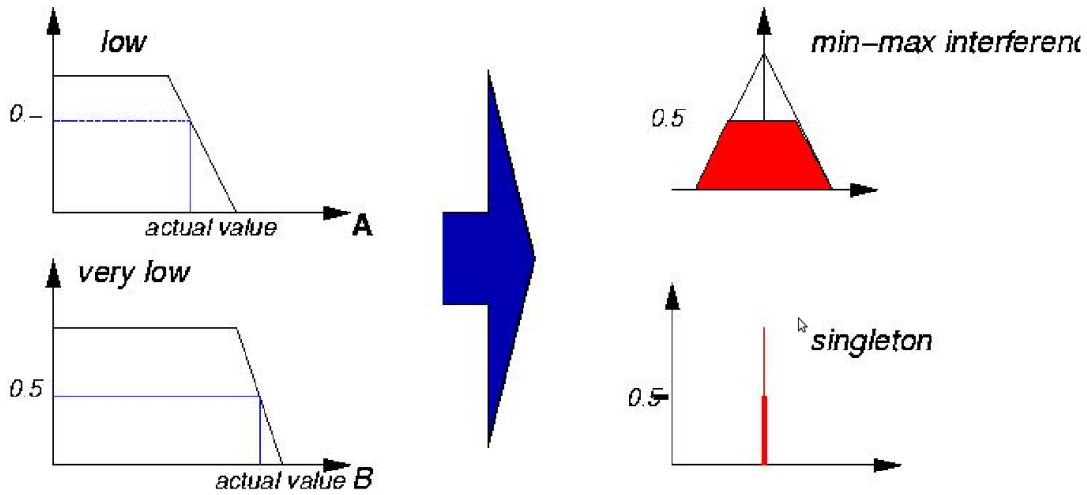


Figura 5.7: Interferenza per la regola IF H very low AND A low THEN Class = class 1

Gli output fuzzy per tutte le regole sono infine aggregati in un singolo insieme fuzzy. Per ottenere una decisione da questo output fuzzy, bisogna procedere ad una defuzzyficazione. Dunque, dobbiamo scegliere un valore rappresentativo come output finale. Esistono molti metodi euristici, uno di questi consiste nel

prendere il centro di massa dell'insieme fuzzy come mostrato in figura.

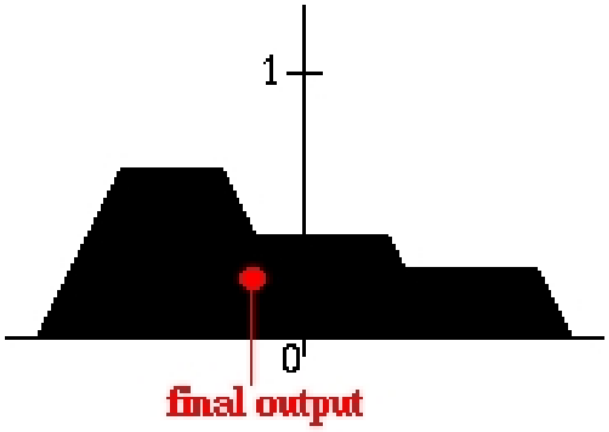


Figura 5.8: Defuzzyficazione effettuata sfruttando il centro di massa

5.3 Motivazioni

La logica fuzzy mette a disposizione un approccio differente al problema del controllo o della classificazione. Questo metodo si concentra su cosa il sistema dovrebbe fare, invece che sul tentativo di modellizzarne il funzionamento. Ci si può concentrare sulla risoluzione del problema, piuttosto che sulla sua formalizzazione matematica, posto che questa sia possibile.

D'altro canto, l'approccio fuzzy richiede un livello di conoscenza sufficiente per la formulazione delle regole e le asserzioni di base, la combinazione di insiemi e la defuzzyficazione.

In generale, l'impiego di logica fuzzy potrebbe risultare utile per processi complessi, per i quali non si disponga di un modello matematico sufficientemente accurato (ad esempio problemi di inversione), per processi non lineari, ecc. Essendo l'interazione uomo-robot riconducibile a tale classe di problemi, l'adozione di una logica fuzzy per la gestione delle interazione e l'interpretazione dei dati risulta certamente una scelta convincente, e finalizzata all'ottenimento di comportamenti il più possibile simili a quelli umani.

Capitolo 6

L'Implementazione (Parte II)

Come concluso dalla precedente sezione, il metodo più efficiente per la gestione dei comportamenti del robot è la modellizzazione dei dati secondo insiemi fuzzy. Ed è quì che ci viene incontro Mr. Brian.

Mr.Brian si è rivelato uno strumento fondamentale nel processo di progettazione: la sua modularità ha permesso la modifica di comportamenti preesistenti evitando di mettere mano al codice di gestione, e operando unicamente a livello logico.

6.1 Fuzzyficazione in Brian

Come già menzionato nella sezione relativa alla teoria sottostante la logica fuzzy, il processo di fuzzyficazione converte le variabili di ingresso in variabili caratterizzate dal loro grado di appartenenza a un insieme. Tale conversione da grandezze deterministiche a fuzzy viene effettuata attraverso le funzioni di appartenenza predefinite per quelle classi. Per utilizzare tale meccanismo è necessario specificare i valori di input e le tipologie di variabili fuzzy ad essi associate. Ad esempio:

```
(PersonDistance DISTANCE)
```

dichiara che il dato PersonDistance ricavato dai sensore è di tipo *DISTANCE*. Il tipo di dato fuzzy *DISTANCE* può essere specificato come segue:

```
(DISTANCE
(SNG (OUT_OF_RANGE 0))
(TRA (IN_CONTACT 1 1 40 40))
```

```
(TRA (VERY_CLOSE 35 40 70 110))
(TRA (CLOSE 70 110 130 190))
(TRI (NEAR 130 190 250))
(TOR (FAR 190 250))
(TOR (VERY_FAR 250 650))
)
```

FIGURA

6.2 Ruolo dei predicati

Partendo dalla definizione e dalla tipologia di variabili fuzzy, si è pronti alla creazione di predicati, ossia di letterali a cui si associa un valore di verità tra 0 e 1. Il predicato:

```
PersonNear = {D Person NEAR}
```

indica, ad esempio, che la variabile *PersonNear* avrà un valore di verità ricavato dal grado di appartenenza del dato *PersonDistance* all'interno dell'insieme NEAR creato in precedenza.

6.3 Comportamenti

L'attivazione dei comportamenti in Brian è subordinata alla verifica della possibilità di esecuzione degli stessi; tale verifica viene effettuata basandosi sulle condizioni definite nel file *cando.ini*. Ad ogni comportamento è associato un predicato attivabile in base ai valori delle variabili su cui opera. Ad esempio:

```
LookForPerson = (AND (AND (P Always)(P OpModeAuto))
                                    (P PersonOutOfRange));
```

indica che il comportamento di ricerca di persone si attiverà solamente qualora il robot si trovi in modalità automatica e non abbia alcuna persona nel campo visivo.

6.4 Modifiche e aggiunte ai comportamenti

Rispetto alla versione precedentemente implementata, i comportamenti sono stati variati per garantire maggior precisione del riconoscimento e, contemporaneamente, minor possibilità di perdere il soggetto agganciato.

Si è procuduti alla limitazione di comportamenti indesiderati, operando una ridefinizione di alcuni parametri descrittivi degli insiemi fuzzy di interesse per l'interazione. Inoltre, le caratteristiche del nuovo sistema di face-tracking hanno permesso la creazione e l'utilizzo di nuovi dati sull'immagine, utilizzati per irrobustire le dinamiche di interazione di E-2?.

6.4.1 Ridefinizione dei fuzzy set

Il primo approccio adottato è stato quello di ridefinire i valori dell'insieme fuzzy riguardante la distanza della persona, misurata dai tre sonar frontali. Partendo dal set di partenza:

```
(DISTANCE
(SNG (OUT_OF_RANGE 0))
(TRA (IN_CONTACT 1 9 40 40))
(TRA (VERY_CLOSE 1 1 70 110))
(TRA (CLOSE 70 110 130 190))
(TRI (NEAR 130 190 250))
(TOR (FAR 190 250))
(TOR (VERY_FAR 250 650))
)
```

sono stati modificati i valori per rendere il robot meno incline a comportamenti errati in prossimità delle persone. Nello specifico, si è tentato di limitare la perdita del soggetto dovuta ad un'eccessiva vicinanza del robot, variando i parametri nella maniera seguente:

```
(DISTANCE
(SNG (OUT_OF_RANGE 0))
(TRA (IN_CONTACT 1 1 40 40))
(TRA (VERY_CLOSE 40 60 90 120))
(TRA (CLOSE 90 120 150 210))
(TRI (NEAR 150 210 270))
(TOR (FAR 210 270))
(TOR (VERY_FAR 270 650))
)
```

Di seguito sono riportati i due grafici di confronto tra i precedenti set fuzzy.

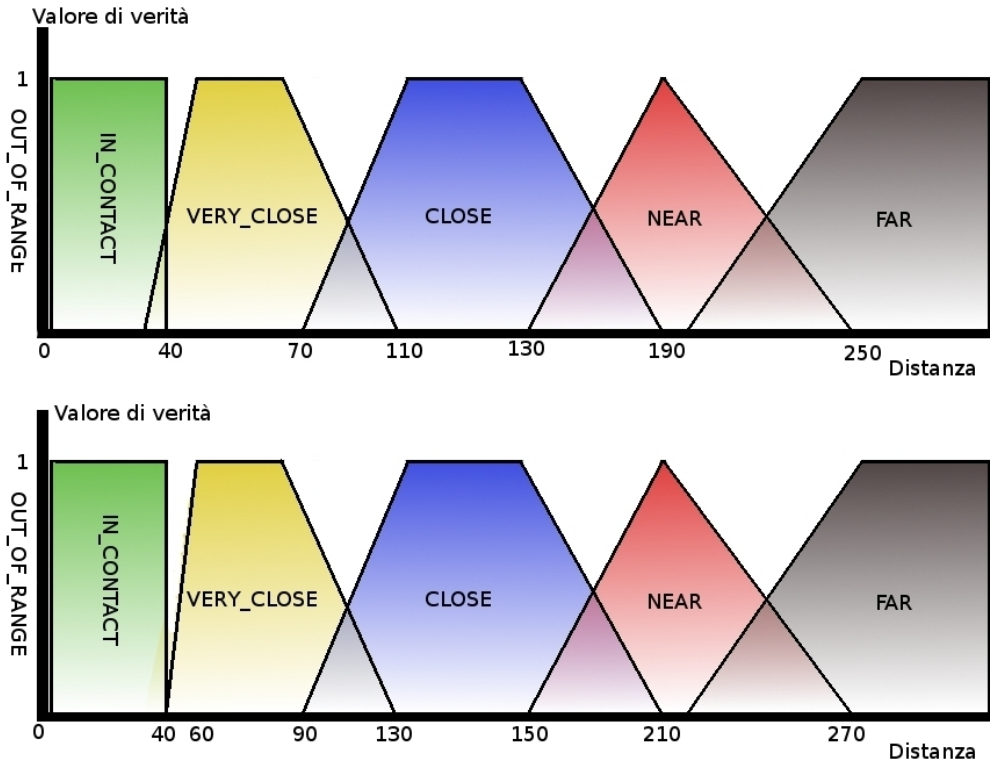


Figura 6.1: Modifiche al set descrittivo della variabile **PersonDistance**

6.4.2 Miglioramento delle procedure di avvicinamento

Per il calcolo della distanza tra robot e soggetto, l'esperto di visione inizialmente implementato si basava unicamente sulla misura dei dati di distanza provenienti dai tre sonar frontali.

Pur risultando sufficientemente accurato nella maggior parte dei casi, si è preferito aggiungere un controllo sulle dimensioni della faccia del soggetto, per poter meglio gestire le dinamiche di avvicinamento ed evitare imrovvise accelerazioni in prossimità del target.

Al codice originale di VisionFrontalExpert è stata apportata la seguente modifica:

```
AddAttribute( FACE_RADIUS, faceDetect->GetRadius(), 1.0 );
```

che permette di aggiungere al messaggio originale un'informazione riguardante il raggio della faccia rilevata dalla telecamera. In base ai test effettuati, si è giunti alla definizione del seguente insieme fuzzy per le dimensioni del viso nell'immagine:

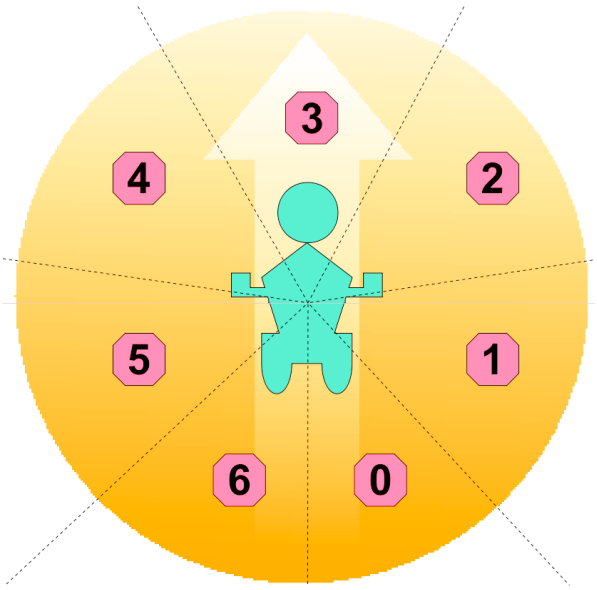


Figura 6.2: Richiamo alla disposizione dei Sonar nel robot

```
(FACE_RADIUS
(SNG (NO_HEAD 0))
(TRA (H_SMALL 1 30 50 80))
(TRA (H_MEDIUM 60 90 110 140))
(TRA (H_BIG 110 150 180 200))
(TOR (H_TOOBIG 180 200))
)
```

a cui corrisponde il seguente diagramma:

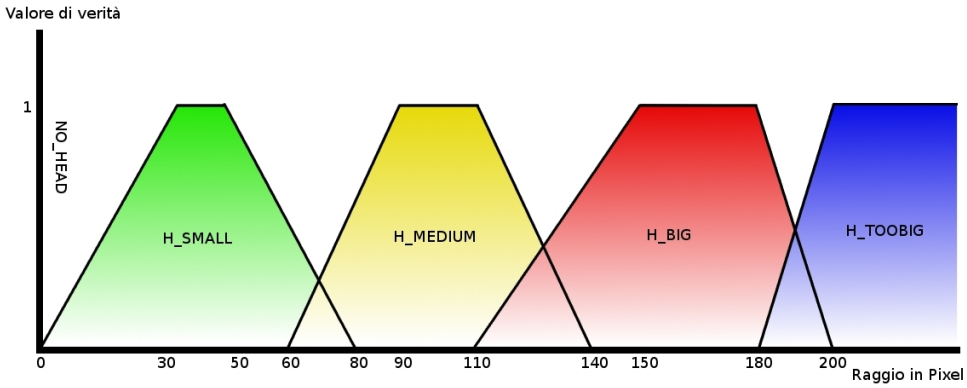


Figura 6.3: Insieme di definizione del raggio della testa

A livello di regole di comportamento, sono state aggiunti i predicati:

```
FaceSmall  = (D HeadRadius H_SMALL)
FaceMedium = (D HeadRadius H_MED)
FaceBig    = (D HeadRadius H_BIG)
FaceTooBig = (D HeadRadius H_TOOBIG)
```

che vengono utilizzati all'interno di GoToPerson.rul per evitare avvicinamenti che causerebbero una perdita della faccia dovuta ad eccessiva prossimità con il soggetto. Ad esempio, l'istruzione:

```
(AND (AND (PersonVeryClose) (PersonN)) (NOT(FaceTooBig))) =>
                                (cmd_w STEADY)(cmd_v MEDIUMFW);
```

tiene conto, nell'applicare la velocità tangenziale, del fatto che la faccia visualizzata non sia eccessivamente grande; la velocità applicata potrà allora essere più sostenuta rispetto al caso in cui la dimensione della faccia rientri nel range fuzzy di FaceTooBig; in questo caso, l'istruzione attivatà sarà:

```
(AND (AND (PersonVeryClose) (PersonN)) (FaceTooBig)) =>
                              (cmd_w STEADY)(cmd_v STEADY);
```

ed il robot rimarrà fermo per evitare di perdere il contatto con il soggetto.

6.4.3 Variabili aggiuntive

Una delle nuove funzionalità del face detector sviluppato consiste nel valutare l'orientamento del soggetto nello spazio tridimensionale. Da un punto di vista strettamente matematico, l'orientamento è definito mediante una terna di misure angolari, che vengono comunemente identificate come angoli di Roll (Rollio), Pitch (Beccheggio) e Yaw (Imbardata).

IMMAGINE

All'interno del codice dell'Esperto di Visione, il messaggio inviato a Mr.Brian è stato arricchito delle tre variabile rappresentanti i tre valori angolari:

```
AddAttribute( FACE_RADIUS, faceDetect->GetRadius(), 1.0 );
AddAttribute( YAW_ANGLE, yaw, 1.0 );
AddAttribute( ROLL_ANGLE, roll, 1.0 );
AddAttribute( PITCH_ANGLE, pitch, 1.0 );
```

I range di valori per gli angoli vanno da -180Â° a 180Â°, e seguono la convenzione della mano destra per il segno. I fuzzy set risultanti sono:

```
(YAW_ANGLE
(TOL (YAW_OVER90_R -120 -90))
(TRI (YAW_90_R -120 -90 -60))
(TRI (YAW_SLIGHT_R -75 -45 -15))
(TRA (YAW_CENTER -30 -15 15 30))
(TRI (YAW_SLIGHT_L 15 45 75))
(TRI (YAW_90_L 60 90 120))
(TOR (YAW_OVER90_L 90 120))
)
```

per l'angolo di Yaw

```
(PITCH_ANGLE
(TOL (PITCH_OVER90_DOWN -120 -90))
(TRI (PITCH_90_DOWN -120 -90 -60))
(TRI (PITCH_SLIGHT_DOWN -75 -45 -15))
(TRA (PITCH_CENTER -30 -15 15 30))
(TRI (PITCH_SLIGHT_UP 15 45 75))
(TRI (PITCH_90_UP 60 90 120))
(TOR (PITCH_OVER90_UP 90 120))
)
```

per l'angolo di Pitch, ed infine

```
(ROLL_ANGLE
(TOL (ROLL_OVER90_R -120 -90))
(TRI (ROLL_90_R -120 -90 -60))
(TRI (ROLL_SLIGHT_R -75 -45 -15))
(TRA (ROLL_CENTER -30 -15 15 30))
(TRI (ROLL_SLIGHT_L 15 45 75))
(TRI (ROLL_90_L 60 90 120))
(TOR (ROLL_OVER90_L 90 120))
)
```

per l'angolo di Roll. Di seguito sono riportati i grafici degli insiemi fuzzy

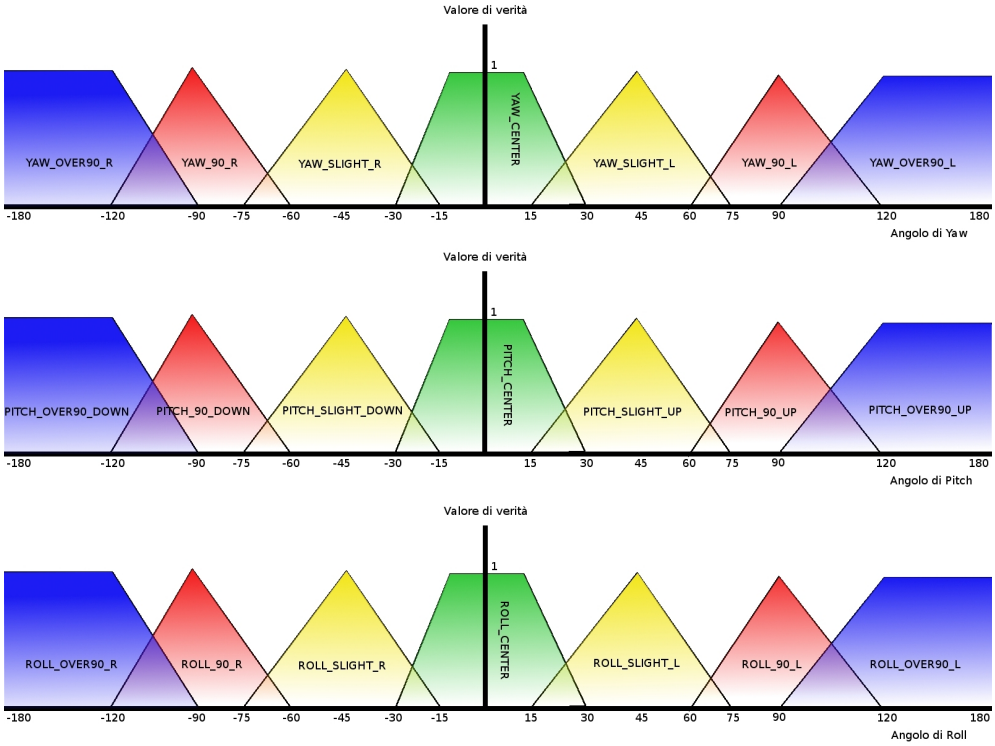


Figura 6.4: Nell'ordine, set fuzzy relativi ad angolo di Yaw, Pich e Roll

Capitolo 7

Conclusioni

In base alle prove effettuate in laboraorio, i miglioramenti rispetto alla versione dell'algoritmo precedentemente utilizzata sono risultati tangibili, ed è emerso come la nuova versione del software sia particolarmente efficace per applicazioni real-time.

All'atto del suo utilizzo sul robot, tuttavia, sono apparsi evdidenti alcuni limiti derivanti dall'architettura stessa di E-2?, sui quali riteniamo bisogni concentrare gli sforzi futuri. A nostro parere, apportare ulteriori modifiche al codice dell'algoritmo produrrebbe risultati scarsamente apprezzabili dal punto di vista applicativo. Ci si dovrebbe piuttosto concentrare sui punti seguenti:

- sviluppo di comportamenti sufficientemente robusti e sofisticati da permettere di evitare le situazioni di errore più critiche
- creazione ex-novo di classificatori di Haar, mirati al riconoscimento dell'orientamento rispetto agli angoli di Yaw (per una migliore individuazione della posa iniziale) e di Pitch (per avere una maggiore libertà nel posizionamento della camera)
- miglioramenti a livello hardware

Per quanto riguarda il primo punto, si tratta di utilizzare i dati acquisiti dalla camera (e messi a disposizone dall'algoritmo) al fine di elaborare regole di comportamento mirate a impedire al robot azioni che potrebbero compromettere la corretta individuazione delle persone. È stato precedentemente discusso l'approccio basato sulla dimensione dei volti, a questo potrebbero essere accostati comportamenti che tengano in considerazione la posizione del viso all'interno del frame (ad esempio per effettuare una rotazione dell'angolo di beccheggio del collo

del robot per la corretta individuazione del soggetto), oppure utilizzino i dati di Yaw, Pitch e Roll per valutare l'approccio migliore al target.

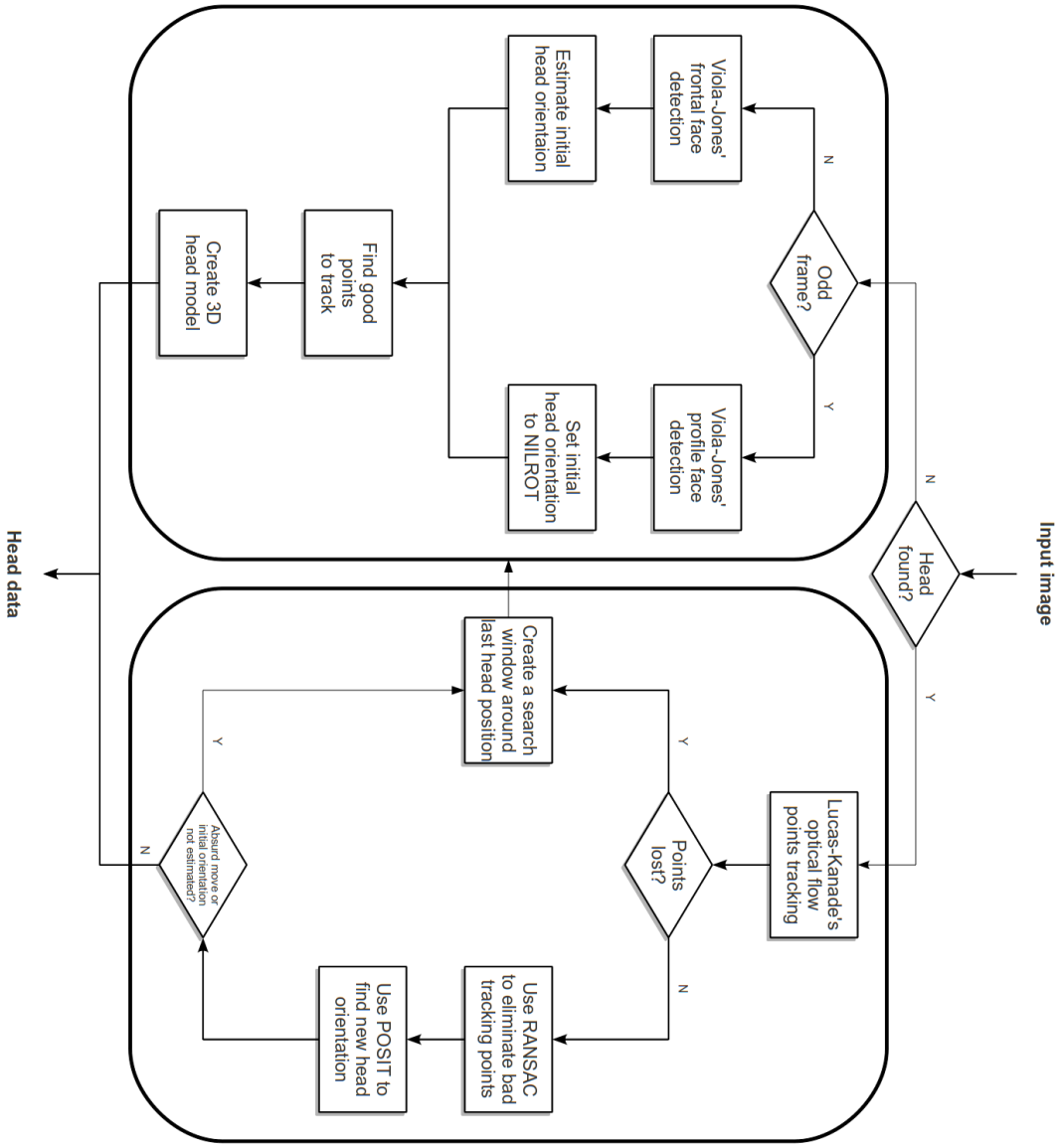
Relativamente allo sviluppo di nuovi classificatori, la riflessione è sorta valutando il contesto di interazione robot-bambino: essendo le differenza di altezza notevoli, e ipotizzando di posizionare la videocamera tra gli occhi del robot, si renderebbe necessario l'adozione di nuovi classificatori, allenati nel riconoscimento di volti con angoli di Pitch non necessariamente frontali.

Infine, ci sentiamo di consigliare una serie di modifiche all'hardware del robot per renderlo in grado di acquisire dati con maggior frequenza e precisione: telecamere con obiettivi grandangolari per la visione, adozione di standard di comunicazione che permettano di gestire tutti i sonar contemporaneamente. Oltre a ciò servirebbe un miglior controllo delle velocità degli attuatori delle ruote, così da rendere i movimenti più fluidi (e naturali) ed eliminare oscillazioni indesiderate.

In conclusione, ci sentiamo di poter affermare di aver implementato un algoritmo di face-tracking robusto, affidabile e preciso, e siamo fiduciosi che il lavoro svolto possa aprire nuove possibilità per lo sviluppo futuro del robot.

Appendice A

Diagramma completo algoritmo



Appendice B

Listato (Parte I)

L'header file della classe: *FaceDetecor.h*

```
1  #ifndef FACEDETECTOR_H_
2  #define FACEDETECTOR_H_
3
4  #include "cv.h"
5  #include "highgui.h"
6
7  using namespace std;
8  using namespace cv;
9
10 #define PI 3.14159265
11
12 #define MIN_POINTS 50
13 #define MIN_POINTS_POSIT 8
14 #define POINT_PERC 80
15 #define MAX_ANGLE PI/3
16 #define OLD_FACE 75
17 #define MAX_FACE 150
18 #define MAX_LOST 30
19
20 #define RANSAC_SAMPLES 8
21 #define RANSAC_DISTANCE_THRESHOLD 30.0
22 #define RANSAC_ITERATIONS 20
23
24 #define EHCIMODELSCALE 1.0
25 #define EHCIFOCUS 602
26
27 enum { FRONTALE, PROFILE_R, PROFILE_L };
28
29 class FaceDetector
```

```
30 {
31 private:
32   double startMatrix[16];
33   double glPositMatrix[16];
34
35   int radius;
36   Point center;
37   float roll, pitch, yaw;
38
39   Mat gray, prev_gray;
40   IplImage *pyramid, *prev_pyramid;
41   vector<Point3f> modelPoints;
42   vector<Point2f> points, prev_points, ransacDeleted;
43
44   CascadeClassifier frontalCascade;
45   CascadeClassifier profileCascade;
46   CascadeClassifier rEyeCascade;
47   CascadeClassifier lEyeCascade;
48   CascadeClassifier mouthCascade;
49
50 //Altre variabili interne
51 ...
52
53   bool detect( Mat& img, double scale );
54   bool detectFace( Mat& img, double scale, int mode );
55   bool update( Mat& img, double dt );
56   bool correct( Mat& img );
57   void draw( Mat& img );
58
59 //Funzioni interne di supporto
60 ...
61
62 public:
63   FaceDetector();
64   FaceDetector( cv::String frontalCascadeName,
65           cv::String profileCascadeName,
66           cv::String rEyeCascadeName,
67           cv::String lEyeCascadeName,
68           cv::String mouthCascadeName );
69
70   cv::Point getPoint();
71   int getRadius();
72   void getGLOrientation( float& roll, float& pitch, float& yaw );
73   void getGlPositMatrix( double myGlPositMatrix[16] );
74
75   bool faceDetect( unsigned char* img_pointer, double scale, double ↩
```

```
        angle );
};

#endif /* FACEDETECTOR_H_ */
```

L'implementazione della classe: *FaceDetecor.cpp*

```
//Costruttori
...

//Getters
...

bool FaceDetector::faceDetect( unsigned char* img_pointer, double scale,↩
     double angle )
{
  IplImage* newImage;
  newImage = cvCreateImage(cvSize(640,480), IPL_DEPTH_8U, 3);
  newImage->imageData = (char*)img_pointer;
  Mat img( newImage );

  if( !lost || lost > MAX_LOST )
  {
    lost = 0;
    if( !faceFound )
      detect( img, 2 );
    else
      update( img, 0 );
  } else
    correct( img );

  draw( img );
  cvReleaseImage( &newImage );
  cvWaitKey(2);

  if( !lost )
    return faceFound;
  else
    return true;
}

bool FaceDetector::detect( Mat& img, double scale )
{
  if( count%2 == 0 )
    return detectFace( img, scale, FRONTALE );
  if( count%4 == 1 )
```

```
39      return detectFace( img, scale, PROFILE_R );
40    if( count%4 == 3 )
41      return detectFace( img, scale, PROFILE_L );
42    return false;
43  }
44
45  bool FaceDetector::detectFace( Mat& img, double scale, int mode )
46  {
47    count++;
48      CascadeClassifier cascade;
49      modelFound = false;
50
51  //Gestione del model
52  ...
53
54      cvtColor( img, gray, CV_BGR2GRAY );
55
56
57    if( !cascade.empty() )
58    {
59      Rect face = detectBiggestHaar( gray, cascade, Size(40,40), cvRound(↩
            img.cols/scale) );
60
61      if( face.width )
62      {
63        faceFound = true;
64
65        roll = 0.0; pitch = 0.0; yaw = 0.0;
66        modelFound = findOrientation( gray, face, roll, pitch, yaw, mode )↩
              ;
67        initStartMatrix( roll, pitch, yaw );
68
69        if( faceFound )
70        {
71          flags = 0;
72          count = 0;
73          insertNewPoints( gray, face, mode );
74          create3DModel( face );
75          initHeadWidth = face.width;
76          radius = cvRound(face.width/2);
77          center.x = face.x + radius;
78          center.y = face.y + radius;
79        }
80      }
81    }
82
```

```
83    return faceFound;
84  }
85
86  bool FaceDetector::update( Mat& img, double dt )
87  {
88    count++;
89    int changed = 0, notStatus = 0;
90    float totErr = 0.0;
91    IplImage* swap_temp;
92
93    prev_points = points;
94    prev_gray = gray.clone();
95    cvtColor( img, gray, CV_BGR2GRAY );
96    CV_SWAP( prev_pyramid, pyramid, swap_temp );
97
98    ransacDeleted.clear();
99
100   if( !prev_points.empty() )
101     trackOptFlow( prev_gray, gray, prev_pyramid, pyramid,
102             points, prev_points,
103             flags, changed, notStatus, totErr );
104
105   if( count < 2 || ( changed > (POINT_PERC/100)*(int)points.size() && !↩
          notStatus ) )//&& totErr < 250 ) )
106   {
107     CvMatr32f rotation_matrix = new float[9];
108     CvVect32f translation_vector = new float[3];
109
110     if( points.size() >= MIN_POINTS_POSIT )
111     {
112       getPositMatrix( rotation_matrix, translation_vector );
113       updateGLPositMatrix( rotation_matrix, translation_vector );
114     }
115   } else {
116     points = prev_points;
117     gray = prev_gray.clone();
118   }
119
120   float new_roll = 0.0, new_pitch = 0.0, new_yaw = 0.0;
121   if( modelFound )
122     getGLOrientation( new_roll, new_pitch, new_yaw );
123
124   bool old = false;
125   if( (count > MAX_FACE) || (points.size() < MIN_POINTS) || (abs(↩
          new_roll) > MAX_ANGLE) || (abs(new_pitch) > MAX_ANGLE) )
126     lost = 1;
```

```
127   if( count < 2 || changed < 0.8*points.size() )
128     lost = 0;
129
130   if( count < 2)
131     initHeadWidth = initHeadWidth*glPositMatrix[14];
132   else
133     if( !lost )
134     {
135       updatePoint();
136       updateRadius();
137     }
138
139   if( (count > OLD_FACE) && (abs(new_roll) < PI/18) && (abs(new_pitch) <↩
          PI/9) && (abs(new_yaw) < PI/9) )
140     old = true;
141
142   roll = new_roll; pitch = new_pitch; yaw = new_yaw;
143   if( !modelFound || lost || old )
144     faceFound = correct( img );
145
146   return faceFound;
147 }
148
149 bool FaceDetector::correct( Mat& img )
150 {
151   Rect search;
152   Rect detected;
153   Mat gray, grayROI;
154   int newWidth;
155
156   cvtColor( img, gray, CV_BGR2GRAY );
157
158     Point p1( center.x-radius, center.y-radius );
159     Point p2( center.x+radius, center.y+radius );
160
161   if( lost )
162   {
163     lost++;
164     newWidth = 100;
165     resizeSearch( Rect(p1,p2), search, Size( gray.cols, gray.rows ), 1 )↩
          ;
166   } else
167   {
168     newWidth = 60;
169     resizeSearch( Rect(p1,p2), search, Size( gray.cols, gray.rows ), 0.5↩
          );
```

```
170   }
171
172   grayROI = gray(search);
173
174   int mode = FRONTALE;
175   detected = detectBiggestHaar( grayROI, frontalCascade, Size(20, 20), ↩
          newWidth );
176   if( !detected.width && lost )
177   {
178     mode = PROFILE_R;
179     detected = detectBiggestHaar( grayROI, profileCascade, Size(20, 20),↩
              newWidth );
180   }
181   if( !detected.width && lost )
182   {
183     mode = PROFILE_L;
184     flip( grayROI, grayROI, 1 );
185     detected = detectBiggestHaar( grayROI, profileCascade, Size(20, 20),↩
              newWidth );
186     flip( grayROI, grayROI, 1 );
187     detected.x = grayROI.cols - detected.x - detected.width;
188   }
189
190   if( detected.width )
191   {
192     faceFound = true;
193
194     detected = detected + Point( search.x, search.y );
195
196     roll = 0.0; pitch = 0.0; yaw = 0.0;
197     modelFound = findOrientation( gray, detected, roll, pitch, yaw, mode↩
              );
198     initStartMatrix( roll, pitch, yaw );
199
200     if( ( modelFound || lost ) && faceFound )
201     {
202       lost = 0;
203       flags = 0;
204       count = 0;
205       insertNewPoints( gray, detected, mode );
206       create3DModel( detected );
207       initHeadWidth = detected.width;
208       radius = cvRound(detected.width/2);
209       center.x = detected.x + radius;
210       center.y = detected.y + radius;
211     }
```

```
212
213     return true;
214   }
215
216   if( lost )
217     return false;
218
219   return true;
220 }
221
222 //Funzioni interne di supporto
223 ...
```

Le funzioni di analisi d'immagine

```
1 Rect detectBiggestHaar( Mat img, CascadeClassifier cascade, Size ↩
      cascadeSize, int newWidth )
2 {
3   Rect result; Mat imgCopy, Mat newImg; vector<Rect> detections;
4
5   imgCopy = img.clone();
6   equalizeHist( imgCopy, imgCopy );
7
8   double scale = (double)img.cols / (double)newWidth;
9   int newHeight = cvRound( (double)img.rows / scale );
10  resize( imgCopy, newImg, Size( newWidth, newHeight ), 0, 0, ↩
        INTER_LINEAR );
11
12  if( !cascade.empty() )
13    cascade.detectMultiScale( newImg, detections, 1.1, 1, ↩
          CV_HAAR_FIND_BIGGEST_OBJECT, cascadeSize );
14
15  if( !detections.empty() )
16  {
17    result = detections.front();
18    result = Rect( cvRound(result.x*scale), cvRound(result.y*scale), ↩
          cvRound(result.width*scale), cvRound(result.height*scale) );
19  }
20
21  return result;
22 }
23
24 void trackOptFlow( Mat prev_gray, Mat gray, IplImage* prev_pyramid, ↩
      IplImage* pyramid, vector<Point2f>& points, vector<Point2f>& ↩
      prev_points, int& flags, int& changed, int& notStatus, float& totErr↩
       )
```

```
25 {
26   vector<uchar> status; vector<float> err;
27
28   IplImage d_prev_grey = prev_gray, d_grey = gray;
29   IplImage *prev_grey = &d_prev_grey, *grey = &d_grey;
30
31   CvPoint2D32f *old_points, *old_prev_points;
32   float* old_err; char* old_status;
33   CvPoint2D32f pt; pt.x = 0.0; pt.y = 0.0;
34
35   old_prev_points = (CvPoint2D32f*)cvAlloc(points.size()*sizeof(pt));
36   old_points = (CvPoint2D32f*)cvAlloc(points.size()*sizeof(pt));
37   old_err = (float*)cvAlloc(points.size()*sizeof(float));
38   old_status = (char*)cvAlloc(points.size());
39
40   for( int i = 0; i < (int)points.size(); i++ )
41   {
42     old_points[i] = points[i];
43     old_prev_points[i] = prev_points[i];
44   }
45
46   cvCalcOpticalFlowPyrLK( prev_grey, grey, prev_pyramid, pyramid, ↩
       old_prev_points, old_points, prev_points.size(), cvSize(10,10), 3,↩
        old_status, old_err, cvTermCriteria(CV_TERMCRIT_ITER|↩
       CV_TERMCRIT_EPS,20,0.03), flags );
47
48   flags |= CV_LKFLOW_PYR_A_READY;
49
50   for( int i = 0; i < (int)points.size(); i++ )
51   {
52     points[i] = old_points[i];
53     prev_points[i] = old_prev_points[i];
54     status.push_back(old_status[i]);
55     err.push_back(old_err[i]);
56   }
57
58   vector<Point2f> valid_points; vector<Point2f> prev_valid_points;
59   vector<float>::const_iterator e = err.begin();
60   vector<uchar>::const_iterator s = status.begin();
61   vector<Point2f>::const_iterator p = prev_points.begin();
62   for( vector<Point2f>::const_iterator r = points.begin(); r != points.↩
       end(); r++, p++, s++, e++ )
63   {
64     if( !*s )
65     {
66       notStatus++;
```

```
67        continue;
68      }
69      valid_points.push_back( *r );
70      prev_valid_points.push_back( *p );
71
72      double dx = r->x - p->x;
73      double dy = r->y - p->y;
74      if( dx*dx + dy*dy >= 2 )
75        changed++;
76
77      totErr += *e;
78    }
79
80    points = valid_points;
81    prev_points = prev_valid_points;
82
83    totErr = totErr/(float)points.size();
84  }
85
86  int ransac( vector<Point2f> imagePoints, vector<Point3f> objectPoints, ↩
        CvMatr32f rotation_matrix, CvVect32f translation_vector, vector<int↩
       >& outliers )
87  {
88    vector<int> bestSet;
89    CvMatr32f rotationTestMatrix = new float[9];
90    CvVect32f translationTestVector = new float[3];
91    int bestSize = 0;
92
93    for( int i = 0; i < (int)imagePoints.size(); i++ )
94      bestSet.push_back(i);
95
96    for( int k = 0; k < RANSAC_ITERATIONS; k++ )
97    {
98      //only choose first 4 points:
99      vector<int> randomPoints = getRandomSet( imagePoints.size(), ↩
            RANSAC_SAMPLES );
100
101     getMatrixUsingPosit( imagePoints, objectPoints, randomPoints, ↩
            rotationTestMatrix, translationTestVector );
102
103     vector<int> inliers;
104     plot2dModel( rotationTestMatrix, translationTestVector, objectPoints↩
            , imagePoints, inliers );
105
106     if( (int)inliers.size() > bestSize )
107     {
```

```
108         bestSize = inliers.size();
109         bestSet.clear();
110         outliers.clear();
111 
112         int dCount = 0;
113         for( int i = 0; i < (int)inliers.size(); i++ )
114         {
115           while( dCount != inliers[i] )
116           {
117             outliers.push_back(dCount);
118             dCount++;
119           }
120 
121           bestSet.push_back(inliers[i]);
122           dCount++;
123         }
124         while( dCount < (int)imagePoints.size() )
125         {
126           outliers.push_back(dCount);
127           dCount++;
128         }
129       }
130     }
131 
132     delete [] rotationTestMatrix;
133     delete [] translationTestVector;
134 
135     //generate rot and trans matrixes with best inliers
136     //now use the bestSet to retrieve the rotation and translation ↩
          matrixes
137     return getMatrixUsingPosit( imagePoints, objectPoints, bestSet, ↩
          rotation_matrix, translation_vector );
138 }
139 
140 //Funzioni interne del RANSAC
141 ...
```

Appendice C

Listato (Parte II)

Il listato (o solo parti rilevanti di questo, se risulta particolarmente esteso) con l'autodocumentazione relativa.
Codice relativo alla definizone degli insiemi fuzzy

Distanza

```
(DISTANCE
(SNG (OUT_OF_RANGE 0))
(TRA (IN_CONTACT 1 1 40 40))
(TRA (VERY_CLOSE 40 60 90 120))
(TRA (CLOSE 90 120 150 210))
(TRI (NEAR 150 210 270))
(TOR (FAR 210 270))
(TOR (VERY_FAR 270 650))
)
```

Angoli di Rotazione

```
##HEAD ANGLES#################################
##YAW##
(YAW_ANGLE
(TOL (YAW_OVER90_R -120 -90))
(TRI (YAW_90_R -120 -90 -60))
(TRI (YAW_SLIGHT_R -75 -45 -15))
(TRA (YAW_CENTER -30 -15 15 30))
(TRI (YAW_SLIGHT_L 15 45 75))
(TRI (YAW_90_L 60 90 120))
(TOR (YAW_OVER90_L 90 120))
```

```
)

##PITCH##
(PITCH_ANGLE
(TOL (PITCH_OVER90_DOWN -120 -90))
(TRI (PITCH_90_DOWN -120 -90 -60))
(TRI (PITCH_SLIGHT_DOWN -75 -45 -15))
(TRA (PITCH_CENTER -30 -15 15 30))
(TRI (PITCH_SLIGHT_UP 15 45 75))
(TRI (PITCH_90_UP 60 90 120))
(TOR (PITCH_OVER90_UP 90 120))
)

##ROLL##
(ROLL_ANGLE
(TOL (ROLL_OVER90_R -120 -90))
(TRI (ROLL_90_R -120 -90 -60))
(TRI (ROLL_SLIGHT_R -75 -45 -15))
(TRA (ROLL_CENTER -30 -15 15 30))
(TRI (ROLL_SLIGHT_L 15 45 75))
(TRI (ROLL_90_L 60 90 120))
(TOR (ROLL_OVER90_L 90 120))
)
```

Raggio della testa

```
##HEAD RADIUS#################################

(FACE_RADIUS
(SNG (NO_HEAD 0))
(TRA (H_SMALL 1 30 50 80))
(TRA (H_MEDIUM 60 90 110 140))
(TRA (H_BIG 110 150 180 200))
(TOR (H_TOOBIG 180 200))
)
```

Definizione di predicati

```
#-------PREDICATI RAGGIO FACCIA--------------
```

```
FaceSmall = (D HeadRadius H_SMALL);
FaceMedium = (D HeadRadius H_MED);
FaceBig = (D HeadRadius H_BIG);
FaceTooBig = (D HeadRadius H_TOOBIG);

#------PREDICATI ANGOLI TESTA----------------
#YAW

YawCenter = (D HeadYaw YAW_CENTER);
YawSmallR = (D HeadYaw YAW_SLIGHT_R);
YawFarR = (D HeadYaw YAW_90_R);
YawTurnedR = (D HeadYaw YAW_OVER90_R);
YawSmallL = (D HeadYaw YAW_SLIGHT_L);
YawFarL = (D HeadYaw YAW_90_L);
YawTurnedL = (D HeadYaw YAW_OVER90_L);

#PITCH

PitchCenter = (D HeadYaw YAW_CENTER);
PitchSmallUP = (D HeadYaw YAW_SLIGHT_UP);
PitchFarUP = (D HeadYaw YAW_90_UP);
PitchTurnedUP = (D HeadYaw YAW_OVER90_UP);
PitchSmallDOWN = (D HeadYaw YAW_SLIGHT_DOWN);
PitchFarDOWN = (D HeadYaw YAW_90_DOWN);
PitchTurnedDOWN = (D HeadYaw YAW_OVER90_DOWN);

#ROLL

RollCenter = (D HeadRoll ROLL_CENTER);
RollSmallR = (D HeadRoll ROLL_SLIGHT_R);
RollFarR = (D HeadRoll ROLL_90_R);
RollTurnedR = (D HeadRoll ROLL_OVER90_R);
RollSmallL = (D HeadRoll ROLL_SLIGHT_L);
RollFarL = (D HeadRoll ROLL_90_L);
RollTurnedL = (D HeadRoll ROLL_OVER90_L);
```

Implementazione delle nuove regole di avvicniamento (file GoToPerson.rul)

```
##################################
## Section Person Very Close
##################################
(AND (AND(PersonVeryClose) (PersonN)) (NOT(FaceTooBig)))
=> (cmd_w STEADY)(cmd_v MEDIUMFW);
(AND (AND(PersonVeryClose) (PersonNE))  (NOT(FaceTooBig)))
=> (cmd_w FASTRX)(cmd_v MEDIUMFW);
(AND (AND(PersonVeryClose) (PersonENE)) (NOT(FaceTooBig)))
=> (cmd_w FASTRX)(cmd_v SLOWFW);
(AND (AND(PersonVeryClose) (PersonS))  (NOT(FaceTooBig)))
=> (cmd_w STEADY)(cmd_v MEDIUMRW);
(AND (AND(PersonVeryClose) (PersonNW))  (NOT(FaceTooBig)))
=> (cmd_w FASTLX)(cmd_v MEDIUMFW);
(AND (AND(PersonVeryClose) (PersonWNW)) (NOT(FaceTooBig)))
=> (cmd_w FASTLX)(cmd_v SLOWFW);
(AND (AND(PersonVeryClose) (PersonSE))  (NOT(FaceTooBig)))
=> (cmd_w FASTLX)(cmd_v MEDIUMRW);
(AND (AND(PersonVeryClose) (PersonESE)) (NOT(FaceTooBig)))
=> (cmd_w FASTLX)(cmd_v SLOWRW);
(AND (AND(PersonVeryClose) (PersonSW))  (NOT(FaceTooBig)))
=> (cmd_w FASTRX)(cmd_v MEDIUMRW);
(AND (AND(PersonVeryClose) (PersonWSW)) (NOT(FaceTooBig)))
=> (cmd_w FASTRX)(cmd_v SLOWRW);

###########  Aggiunta  ##########
(AND (AND(PersonVeryClose) (PersonN)) (FaceTooBig))
=> (cmd_w STEADY)(cmd_v STEADY);
(AND (AND(PersonVeryClose) (PersonNE))  (FaceTooBig))
=> (cmd_w SLOWRX)(cmd_v SLOWFW);
(AND (AND(PersonVeryClose) (PersonENE)) (FaceTooBig))
=> (cmd_w SLOWRX)(cmd_v SLOWFW);
(AND (AND(PersonVeryClose) (PersonS))  (FaceTooBig))
=> (cmd_w STEADY)(cmd_v SLOWRW);
(AND (AND(PersonVeryClose) (PersonNW))  (FaceTooBig))
=> (cmd_w SLOWLX)(cmd_v SLOWFW);
```

```
(AND (AND(PersonVeryClose) (PersonWNW)) (FaceTooBig))
=> (cmd_w SLOWLX)(cmd_v SLOWFW);
(AND (AND(PersonVeryClose) (PersonSE))  (FaceTooBig))
=> (cmd_w SLOWLX)(cmd_v SLOWRW);
(AND (AND(PersonVeryClose) (PersonESE)) (FaceTooBig))
=> (cmd_w SLOWLX)(cmd_v SLOWRW);
(AND (AND(PersonVeryClose) (PersonSW))  (FaceTooBig))
=> (cmd_w SLOWRX)(cmd_v SLOWRW);
(AND (AND(PersonVeryClose) (PersonWSW)) (FaceTooBig))
=> (cmd_w SLOWRX)(cmd_v SLOWRW);
```

Modifica del codice C++ dell'esperto di visione

```
1 //-------------------------------------------------------------------
2   float yaw,pitch,roll;
3   faceDetect->getGLOrientation(roll, pitch, yaw);
4   AddAttribute(FACE_RADIUS, faceDetect->GetRadius() ,1.0);
5   AddAttribute(YAW_ANGLE,   yaw , 1.0);
6   AddAttribute(ROLL_ANGLE,  roll, 1.0);
7   AddAttribute(PITCH_ANGLE,   pitch,  1.0);
8 //-------------------------------------------------------------------
```

Appendice D

Esempio di utilizzo dell'algoritmo

```
#include "cv.h"
#include "highgui.h"
#include <iostream>
#include <cstdio>
#include "FaceDetector.h"

using namespace std;
using namespace cv;

String frontalCascadeName =
"/usr/local/share/opencv/haarcascades/haarcascade_frontalface_alt2.xml";
String profileCascadeName =
"/usr/local/share/opencv/haarcascades/haarcascade_profileface.xml";
String rEyeCascadeName =
"/usr/local/share/opencv/haarcascades/haarcascade_mcs_righteye.xml";
String lEyeCascadeName =
"/usr/local/share/opencv/haarcascades/haarcascade_mcs_lefteye.xml";
String mouthCascadeName =
"/usr/local/share/opencv/haarcascades/haarcascade_mcs_mouth.xml";

int main( int argc, char** argv )
{
  Mat currentFrame;
  FaceDetector detector;
    double t = 0;

    VideoCapture camera( CV_CAP_ANY );
    if( !camera.isOpened() )
    {
      cout<< "Could not initialize capturing..." <<endl;
```

```
31        exit(-1);
32      }
33
34      detector = FaceDetector( frontalCascadeName,
35                  profileCascadeName,
36                  rEyeCascadeName,
37                  lEyeCascadeName,
38                  mouthCascadeName );
39
40      cvNamedWindow( "result", 1 );
41
42      for(;;)
43      {
44        camera >> currentFrame;
45        flip( currentFrame, currentFrame, 1 );
46        imshow( "result", currentFrame);
47
48        if( waitKey( 2 ) >= 0 )
49          break;
50      }
51
52      for(;;)
53      {
54      t = (double)cvGetTickCount();
55      camera >> currentFrame;
56      flip( currentFrame, currentFrame, 1 );
57
58      detector.faceDetect( currentFrame.data, 0, 0 );
59
60      t = (double)cvGetTickCount() - t;
61      //printf( "update time = %g ms\n", t/((double)cvGetTickFrequency()↩
            *1000.) );
62
63      if( waitKey( 2 ) >= 0 )
64        goto _cleanup_;
65      }
66
67      waitKey(0);
68  _cleanup_:
69
70      cvDestroyWindow("result");
71
72      return 0;
73  }
```