POLITECNICO DI MILANO
Facoltà di Ingegneria dell'Informazione
Corso di Studi in Ingegneria Informatica

Studio e Realizzazione di un Sistema d'Interazione Uomo-Robot

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Relatore: Prof. Andrea Bonarini

Tesi di Laurea Specialistica di:
Cristian Mandelli, matricola 739242
Deborah Zamponi, matricola 739284

Anno Accademico 2010-2011

Alle nostre famiglie...

Sommario

Lo sviluppo delle tecnologie informatiche ed i progressi dell'intelligenza artificiale hanno permesso, negli ultimi anni, notevoli sviluppi nel campo dell'interazione uomo-robot (*Human Robot Interaction - HRI*).
In particolare, unendo tali conoscenze alle possibilità tecniche offerte per il controllo del robot e l'interpretazione dei dati sensoriali, abbiamo implementato un complesso sistema d'interazione su E-2?, un robot da intrattenimento sviluppato dal Politecnico di Milano; ciò è stato possibile ricorrendo a telecamere e sensori sviluppati appositamente per l'interazione uomo-macchina, come ad esempio il Kinect.

I risultati finali mostrano come, l'utilizzo di un dispositivo multisensore come il Kinect, unito allo studio dell'interazione uomo-uomo, abbia permesso di implementare un sistema per il controllo di robot sociali aventi l'obiettivo di riprodurre un'interazione *human-like*, in differenti contesti operativi.

Ringraziamenti

Ringraziamo, prima di tutti, i nostri genitori che in questi anni ci hanno premurosamente accudito e confortato nei momenti critici della carriera universitaria e non solo. Grazie soprattutto per aver sopportato le nostre continue trasferte Busto-Ponte e le conseguenti valigie sempre in giro.

Un grazie particolare alla Lauretta (o "disastro") che in questi ultimi tre anni si è privata, più o meno volentieri, del suo spazio in cameretta per lasciare a noi la possibilità di muoverci il più agevolmente possibile. Grazie anche per tutte le sere in cui ti sei dovuta chiudere in camera a vedere la TV, a seguito della classica frase: "Noi dobbiamo studiare!".

Ringraziamo il Prof. Bonarini per averci seguito passo dopo passo nell'elaborazione del presente progetto e per la disponibilità dimostrataci in ogni occasione in cui qualche intoppo limitava il nostro lavoro.

Un sentito ringraziamento anche ai ragazzi dell'AIRLab e soprattutto a Davide Rizzi, la cui conoscenza ed esperienza su E-2? (e non solo) ci ha permesso in più di un'occasione di "alleggerire la pillola".

Grazie agli amici di sempre: Giacomo, Roberto, i fratelli Segala, Tristano, Massimo ed Antonella, Davide ed Ylenia,... Le svariate occasioni passate in vostra compagnia ci hanno permesso di staccare la spina e ricaricare le pile per affrontare più serenamente i nostri impegni universitari.

Infine, un ringraziamento al destino che ci ha fatto incontrare ed unire; il condividere la vita privata ed universitaria, tra gioie e dolori, ci ha permesso di diventare sempre più uniti. Questo traguardo, raggiunto mano nella mano, sarà una delle pietre miliari della nostra storia!

Indice

Capitolo 1

Introduzione

"Mio padre ha provato ad insegnarmi le emozioni umane. Sono... difficili."

Sonny (dal film - Io, Robot)

Lo sviluppo delle attitudini sociali nei robot ha una storia relativamente recente nell'ambito degli studi di robotica ed intelligenza artificiale e si sta imponendo sempre più come requisito necessario per i robot che devono interagire o collaborare con altri robot o con esseri umani.

La ricerca sulla **Human Robot Interaction (HRI)** ha generato numerosi cambiamenti circa la natura dell'interazione e del comportamento sociale tra robot ed essere umano ed è proprio in questo contesto che l'elaborato qui proposto va a collocarsi.

In particolare, il progetto di seguito presentato si pone l'obiettivo di sviluppare un sistema di analisi del volto umano robusto ed economico, affiancato ad un meccanismo di controllo del robot da esibizione E-2?, con lo scopo di ottenere un significativo livello di interazione uomo-macchina.

Per il conseguimento degli obiettivi prefissati, si è resa necessaria l'analisi di numerose informazioni provenienti da moduli sensori distribuiti sul corpo del robot, quali telecamere IR e sonar. Un importante contributo per l'ottenimento di tali informazioni è stato dato dall'utilizzo del dispositivo Kinect, che si è reso fondamentale per il sistema di visione da noi implementato. Quest'ultimo, in particolare, ha permesso il riconoscimento di diversi stati emotivi dell'interlocutore umano, dando così la possibilità di modificare il comportamento e le espressioni del robot nel corso dell'interazione. A tale scopo, sono stati implementati diversi algoritmi aventi l'obiettivo primario

di fornire dati utili per la stima dello stato di interesse e di coinvolgimento dell'utente. Algoritmi per il riconoscimento del volto, degli occhi e dei movimenti dell'utente, in unione ad informazioni provenienti da moduli sensore quali sonar e Kinect, hanno permesso il raggiungimento degli obiettivi preposti.

L'algoritmo di controllo del robot, invece, utilizza le informazioni precedentemente estratte per attuare determinati comportamenti, quali espressioni del volto e pose, in accordo con il livello di attenzione mostrato dall'interlocutore umano.

I risultati ottenuti dimostrano come gli studi compiuti nel campo della HRI, supportati dalle moderne tecnologie attualmente in commercio, permettono analisi di alto livello mirate allo studio non solo di caratteristiche fisiche, ma anche di caratteristiche emozionali quali stati d'animo ed espressioni.

La tesi è suddivisa in cinque capitoli fondamentali ed alterna aspetti tecnici molto dettagliati (cap. 3,4 e 5) ad altri più discorsivi aventi lo scopo di presentare in modo esaustivo l'ambiente operativo in cui il nostro elaborato andrà a collocarsi (cap. 2) e di presentare i risultati ottenuti (cap. 6).

Capitolo 2 *Stato dell'arte*: vengono presentati i recenti sviluppi ottenuti in merito a tematiche di HRI affiancati agli studi svolti negli ultimi anni circa l'individuazione di feature chiave per la stima dello stato emozionale di un essere umano durante un'interazione.

Capitolo 3 *Tecnologie Hardware e Software*: nella prima parte, vengono esposti tutti i principali componenti hardware del robot E-2? utilizzate per il raggiungimento degli obiettivi del presente elaborato. Nella seconda parte, invece, viene presentato brevemente l'ambiente ROS, utilizzato come software di sviluppo.

Capitolo 4 *Il sistema di visione*: vengono illustrate le principali scelte progettuali effettuate durante lo sviluppo del modulo di visione. Per ogni algoritmo si analizzeranno le problematiche riscontrate, i risultati ottenuti ed alcune parti di codice che hanno permesso l'implementazione delle funzionalità più significative.

Capitolo 5 *Il processo d'interazione*: viene qui analizzato l'intero processo di stima del livello di attenzione ed interesse dell'interlocutore durante le varie fasi dell'interazione. All'interno del capitolo, inoltre, verranno analizzate le primitive di interazione implementate per l'attuazione delle espressioni facciali, delle pose e dei comportamenti del robot.

Capitolo 6 *Verifiche sperimentali*: vengono presentati i risultati ottenuti nei test effettuati per la valutazione della robustezza e delle prestazioni del sistema di visione e di controllo del robot.

Capitolo 2

Stato dell'arte

"Ogni volta che impariamo qualcosa di nuovo,
noi stessi diventiamo qualcosa di nuovo"

Leo Buscaglia

Stiamo vivendo in un'epoca in cui, grazie allo sviluppo delle tecnologie informatiche ed ai progressi nel campo dell'intelligenza artificiale, della simulazione e comprensione del linguaggio verbale e del controllo remoto, si è verificato un significativo avanzamento nel campo della robotica, la quale offre sempre più applicazioni orientate verso l'interazione uomo-robot.

Queste nuove generazioni di robot nascono con l'obiettivo di offrire alle persone un'esperienza piacevole, basata su un'interazione amichevole ed informativa. Il raggiungimento di un'interazione di questo tipo, tuttavia, richiede abilità che vadano oltre la semplice percezione dell'ambiente; il robot, infatti, deve essere in grado di comprendere i comportamenti e gli stati emotivi del suo interlocutore, reagendo ad essi di conseguenza.

Proprio dalla necessità di cogliere le emozioni dell'utente nasce il nostro interesse verso l'interpretazione delle espressioni umane, quale principale canale di informazione circa il livello di interesse (*engagement*) dell'interlocutore.

2.1 HRI: la socialità nei robot

L'interazione uomo-robot (HRI - *Human Robot Interaction)* è un'area di ricerca multidisciplinare in costante sviluppo, ricca di spunti per ricerche

avanzate; essa gioca un ruolo fondamentale nella realizzazione di robot che operano in ambienti aperti e cooperano con gli esseri umani. Obiettivi di questo tipo richiedono lo sviluppo di tecniche che permettano ad utenti inesperti di usare i loro robot in modo semplice e sicuro, utilizzando interfacce intuitive e naturali [8].

Le sue origini sono da ricercarsi nella letteratura fantascientifica e, in particolare, fu grazie ai racconti ed ai romanzi di Isaac Asimov (Figura 2.1) che il termine *robotica* divenne popolare. A lui si deve l'idea delle *tre leggi della robotica*, descritte nel romanzo *Io, Robot* e la previsione di un mondo in cui i robot diventino parte integrante della società e la robotica rappresenti una potente industria.

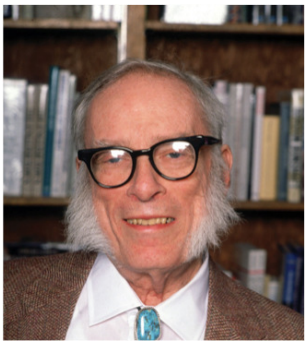

Figura 2.1: Isaac Asimov.

La visione futuristica di Asimov, tuttavia, non si discosta di molto da quanto effettivamente accaduto negli ultimi vent'anni, durante i quali anche il grande pubblico ha considerato sempre più possibile (ed in alcuni casi auspicabile) l'introduzione dei robot nella vita quotidiana degli esseri umani: come partner in ambito lavorativo, come ausilio nella riabilitazione e assistenza ai disabili, o, addirittura, come compagni di gioco nel tempo libero.

I tentativi di creare robot in grado di esibire comportamenti sociali ed interagire con gli essere umani, hanno popolato la storia della recente robotica. La ricerca nel settore si è rapidamente estesa dalla progettazione di robot ispirati alle caratteristiche biologiche e comportamentali di organismi animali a quella di robot sociali ispirati alle modalità di relazione e comunicazione degli esseri umani (Figura 2.2).

Focalizzando l'attenzione sullo sviluppo dei robot sociali, numerosi ricercatori hanno evidenziato l'importanza, in tal contesto, di eseguire uno studio approfondito sia dal punto di vista della teoria dell'interazione che dal punto

Figura 2.2: L'evoluzione dalle tartarughe robotiche di Walter (1940) alla nascita dei geminoidi(2007).

di vista della progettazione del robot. La ricercatrice P.Marti [23], in particolare, ha evidenziato tre aspetti che è necessario tenere in considerazione per lo sviluppo di questa tipologia di robot.

Un primo elemento rilevante è che gli esseri umani tendono a percepire i robot in modo diverso dalle altre macchine ed una ragione è sicuramente legata alla loro "fisicità". Nell'interazione con il computer, infatti, la modalità input/output legata a tastiera e schermo ed alla posizione dell'utente quasi esclusivamente seduta, limitano drasticamente le possibilità di interazione. La fisicità e la tangibilità dei robot autonomi, invece, funzionano da catalizzatori dell'interazione e creano quel particolare fenomeno per cui l'essere umano tende ad antropomorfizzare i robot, anche nei casi in cui il loro aspetto fisico sia chiaramente distante da quello di un essere umano.

Un secondo fattore d'interesse è legato alla mobilità del robot. Si è infatti in presenza di macchine che percepiscono le caratteristiche fisiche dell'ambiente che li circonda e che negoziano l'uso delle risorse presenti nello spazio con altri oggetti, robot o esseri umani. L'attenzione è dunque rivolta alla presenza di comportamenti dinamici che si modificano in relazione alle condizioni ambientali che via via si creano.

Un terzo aspetto di rilievo è quello relativo alla capacità dei robot autonomi di prendere decisioni. Benché tale abilità sia presente anche in altri sistemi come gli agenti software, essa assume una connotazione particolare nei robot, dove tale capacità è combinata con quella di relazionarsi con l'ambiente, di negoziare l'uso delle risorse a disposizione in modo dinamico e di possedere componenti fisiche profondamente integrate con quelle funzionali.

2.1.1 Studi teorici sulla socialità dei robot

Questi tre aspetti ben si rispecchiano negli studi di Breazeal [11], la quale ha fornito una classificazione dei robot sociali, sulla base sia delle capacità del robot di supportare il modello sociale che gli è stato attribuito, sia della complessità dello scenario d'interazione che il robot è in grado di gestire. In particolare, Breazeal ha individuato le seguenti categorie di robot sociali:

- **Suggestivi (socially evocative):** sono robot che hanno fatto affidamento sulla propensione umana ad antropomorfizzare e capitalizzare le sensazioni provate, nel momento in cui allevano, si prendono cura o sono coinvolti dalla loro "creazione";

- **Collocati (socially situated):** sono robot circondati da un ambiente sociale che sono in grado di percepire ed a cui sono in grado di reagire. I robot di questa categoria sono in grado di distinguere, all'interno dell'ambiente, gli esseri umani da qualsiasi altro oggetto;

- **Socievoli (sociable):** sono robot che, di propria iniziativa, entrano in contatto con gli essere umani con l'obiettivo di raggiungere i loro scopi sociali (guidarli, suscitare emozioni, etc.). Questi robot richiedono l'implementazione di un modello complesso delle competenze ed attitudini sociali;

- **Intelligenti (socially intelligent):** sono robot che manifestano caratteristiche dell'interazione sociale tipica dell'uomo, basandosi su modelli della cognizione umana e delle competenze sociali.

A queste quattro classi, Fong [33] ed i suoi ricercatori hanno deciso di aggiungere un'ulteriore categoria di robot, denominata *robot interattivi (Socially Interactive Robots)*, entro la quale vanno ad inserire quella tipologia di robot per cui l'interazione sociale gioca un ruolo fondamentale, distinguendo così questi robot da quelli caratterizzati da un'interazione uomo-robot convenzionale, come quelli utilizzati per gli scenari di teleoperazione.

Ma quali sono le abilità sociali che un robot deve avere? A tal proposito Dautenhanh [16] sottolinea come, data la grande varietà di domini applicativi possibili o attuali, la risposta a questa domanda dipenda proprio dal particolare dominio applicativo.
Nella Figura 2.3 è riportata la lista redatta da Dautenhahn dei diversi domini applicativi, in ordine crescente rispetto alle abilità sociali richieste dal robot: si può osservare in cima alla lista la presenza, ad esempio, dei robot

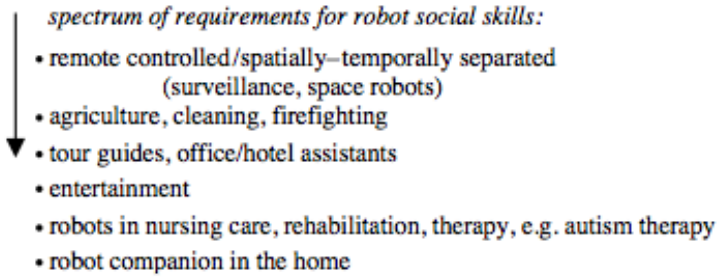

Figura 2.3: Elenco dei possibili domini applicativi di robot sociali, in ordine crescente rispetto alle abilità richieste.

che operano nello spazio (Figura 2.4), i quali non necessitano di particolari abilità sociali, ma solo della capacità di collaborare con altri robot.

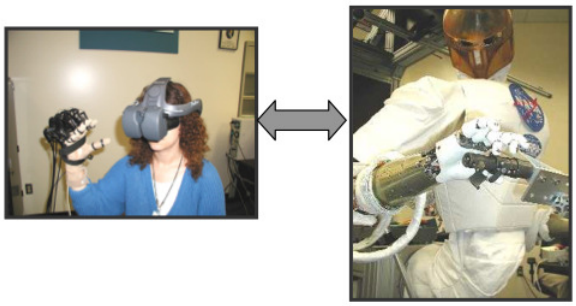

Figura 2.4: Esempio di teleoperazione con il robot ROBONAUT della NASA.

In opposizione a questi, un robot che consegna la posta all'interno di un ufficio ha un contatto quotidiano con i clienti e quindi una corretta definizione dei campi di applicazione, con le relative abilità sociali richieste, contribuirà a rendere più adeguata l'interazione con le persone.

All'ultimo posto della lista troviamo, invece, i robot utilizzati come compagni domestici, per le persone anziane o per assistere disabili (Figura 2.5); questi robot devono poter gestire una grande varietà di abilità sociali perché siano accettabili per l'essere umano.
Se tali abilità non fossero implementate correttamente, alcuni robot potrebbero addirittura non essere utilizzati e quindi venir meno al loro compito di assistente.

Al fine di decidere quali abilità sociali siano richieste per un robot, dunque, è importante attuare un'analisi dettagliata di diversi aspetti, quali il

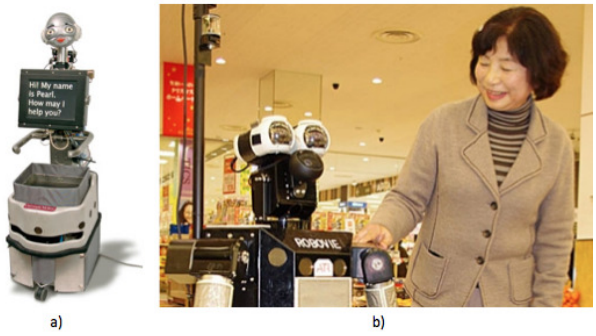

Figura 2.5: Esempi di robot utilizzati nell'assistenza agli anziani: Pearl a sinistra e Robovie a destra.

dominio applicativo, la natura e la frequenza con cui il robot entrerà in contatto con l'uomo, etc.

Tutti questi fattori sono considerati da Dautenhahn dei criteri di valutazione indispensabili per la realizzazione di un robot adeguato ai suoi fini e, in relazione a ciò, la ricercatrice propone una lista (Figura 2.6) contenente le principali caratteristiche che, a suo avviso, bisogna tenere in considerazione.

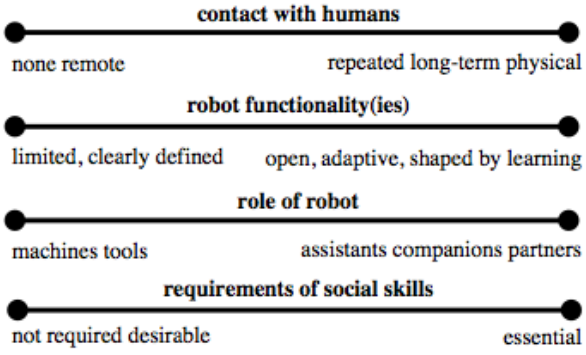

Figura 2.6: I criteri di valutazione da utilizzare per identificare le abilità sociali richieste ad un robot.

Si può osservare come la Dautenhahn abbia definito per ciascuna variabile in gioco (contatto con gli umani, funzionalità del robot, etc) un range di valori possibili che, connessi uno all'altro, riescono a rendere l'idea del livello di abilità sociale richiesta per il robot che si sta studiando.

2.1.2 Robot sociali per il grande pubblico

I robot con comportamento sociale sono spesso progettati ad imitazione di esseri umani ed animali (*life-like* robot) sia nelle loro caratteristiche fisiche che in quelle comportamentali, cognitive ed emotive.

La somiglianza dei robot con gli esseri umani o con animali crea inevitabilmente grandi aspettative nell'interlocutore umano che spesso rimangono frustrate dall'interazione. Un robot che sia dotato di linguaggio naturale, ad esempio, può facilmente diventare un interlocutore noioso o irritante, nel caso in cui non riesca a sostenere un adeguato livello di complessità nella comunicazione [23].

Norman [25] a questo proposito suggerisce di progettare robot che manifestino il loro essere "sistemi imperfetti" e che esibiscano con chiarezza i loro limiti, risultando quindi poi non più così simili all'essere umano che tentano di imitare. Numerosi studi, tuttavia, hanno evidenziato come interazioni con robot di questo tipo siano risultate particolarmente fallimentari.

Si consideri, ad esempio, il cane-robot *Sony Aibo* (Figura 2.7) sviluppato dalla Sony e lanciato sul mercato nel 1999. Sony Aibo è in grado di rispondere a comandi vocali e di interagire con esseri umani; si muove in modo autonomo nello spazio e può esprimere alcuni stati emotivi.

Figura 2.7: Il cane-robot Sony Aibo.

Come molti robot della sua categoria, però, Sony Aibo è totalmente privo di caratteristiche fisico percettive legate all'impressione tattile con la superficie del suo corpo, che risulta essere fredda, di materiale plastico e non particolarmente piacevole da accarezzare.

Tale problematica è stata ben sottolineata dall'esperimento di alcuni ricercatori che prevedeva l'utilizzo di Sony Aibo con un gruppo di anziani: i ricercatori, infatti, hanno dovuto vestire il robot per facilitare un'interazione che si è rivelata da subito piuttosto problematica.

Il motivo della tiepida accettazione di Sony Aibo, così come di altri robot che, come lui, si manifestano estremamente verosimili nell'aspetto fisico e sofisticati nel comportamento, va ricercato in quello che Masahiro Mori ha descritto come l'effetto *uncanny valley* (Figura 2.8). In particolare, Mori ha osservato che quanto più verosimile è il robot nel suo aspetto fisico e comportamentale ad un essere umano o animale, tanto più esso diventa familiare (o credibile) per l'essere umano. A questo fenomeno, però, si contrappone la brusca caduta di familiarità che si manifesta quando il robot è così simile da poter essere scambiato per reale. Ecco dunque che il robot genera il cosiddetto effetto *uncanny valley*, ossia un sentimento di frustrazione da aspettativa inattesa [23].

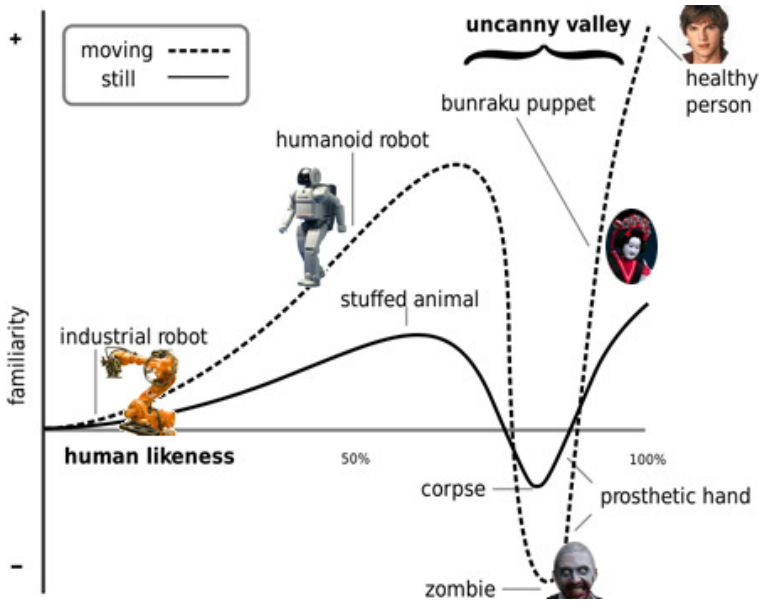

Figura 2.8: L'ipotetica risposta emotiva di un essere umano è qui confrontata con il livello di antropomorfismo di un robot. L'uncanny valley, come evidente dal grafico, è la regione caratterizzata da una risposta emotiva negativa rispetto a robot human-like. Dal grafico è evidente, inoltre, come anche le capacità di movimento del robot amplifichino la risposta emotiva dell'essere umano.

Per ovviare a questa tipologia di problematiche, Takahori Shibata ha

pensato di progettare e realizzare un robot che non solo rispecchiasse le caratteristiche morfologiche e comportamentali dell'animale a cui era ispirato, ma che diventasse anche un catalizzatore di comunicazione emotiva, ossia un animale robot che, per sua natura, stimolasse sentimenti quali il prendersi cura, affetto, tenerezza e mansuetudine. È così nato *Paro* (Figura 2.9), un cucciolo di foca appena nata, coperta di soffice pelliccia e dotata di movimenti e versi tipici degli esemplari giovanissimi.

Figura 2.9: Il robot Paro.

Grazie alla sofisticata rete di sensori posti al di sotto della sua pelliccia, Paro è in grado di reagire alle carezze ed alle sollecitazioni esterne, muovendo in modo coordinato il corpo e la testa, sbattendo gli occhi, emettendo suoni e facendo le fusa se abbracciato. Paro, inoltre, ha una propria vita fisiologica: quando le sue batterie sono cariche è più vivace, ma se lavora per molto tempo, manifesta stanchezza ed i suoi movimenti rallentano. È stata quindi progettata, per questo robot, anche una simulazione del ciclo circadiano e tale innovazione ha riscontrato da subito molto successo sull'interlocutore umano.

Paro sembra dunque avere tutte le caratteristiche necessarie per un robot sociale adatto al grande pubblico; le sue qualità di robot sociale, infatti, l'hanno portato ad essere utilizzato anche in ambito terapeutico per bambini affetti da deficit cognitivi, sensoriali e motori. Le sperimentazioni avvenute in tale contesto hanno manifestato la totale estraneità di Paro all'effetto *uncanny valley*: le sue caratteristiche morfologiche insieme all'impressione tattile che si riceve dal contatto fisico, stimolano l'esplorazione del corpo del robot e delle sue caratteristiche comportamentali.

A fronte degli studi analizzati, si può dunque affermare che il successo di un robot sociale sia legato alla sua capacità di far percepire l'esperienza d'interazione non soltanto a livello fisico e funzionale, ma anche esteti-

co, percettivo ed emotivo; un robot, infatti, riscuote maggiore successo se progettato come mediatore di esperienza, piuttosto che come replica delle capacità espressivo-emotive umane o animali.

2.2 HRI: gli studi sull'interazione

L'incremento delle capacità dei robot e delle loro abilità a svolgere sempre più compiti in modo autonomo, ha generato la necessità di studiare in modo approfondito l'interazione che un essere umano potrebbe instaurare con essi.

Due enti statunitensi, la *National Science Foundation (NSF)* ed il *Department of Energy (DOE)*, hanno definito l'interazione uomo-robot di importanza strategica, sottolineando come ciò sia dovuto soprattutto ai recenti sviluppi tecnologici che hanno permesso ai robot di uscire dalle mura delle fabbriche ed entrare all'interno delle mura domestiche.

Le questioni legate alle interfacce e all'interazione con l'uomo sono da molto tempo oggetto della ricerca nell'ambito della robotica: tipicamente, le persone che lavorano con i robot hanno mansioni di supervisionamento e/o teleoperazione, quindi i primi studi svolti al miglioramento delle interfacce sono stati guidati dalla necessità di migliorare questo tipo di interazione.

In generale, si può affermare che, fino a poco tempo fa, l'attenzione della comunità robotica è stata principalmente robot-centrica, ossia caratterizzata da una maggiore enfasi sulla sfida tecnologica per ottenere controllo e mobilità intelligenti. Solo di recente alla visione robot-centrica si è contrapposta la visione uomo-centrica, ossia l'interesse verso lo sviluppo, nei robot, di un'intelligenza di tipo umano che li rendesse in grado di interagire con gli essere umani nella maniera in cui essi comunicano tra loro. Questa seconda visione va dunque ad enfatizzare lo studio degli esseri umani come modelli per i robot [8].

2.2.1 I modelli d'interazione

Per ottenere un'interazione uomo-robot che sia efficiente ed efficace, si è resa fondamentale la definizione di modelli che esaminassero le tipologie di interazione necessarie tra uomo e robot. Uno dei primi modelli elaborati fu il cosiddetto *modello di Norman* [15], che identifica le fasi principali

di cui si compone l'interazione, fornendo una valida struttura logica per la progettazione e la valutazione. Come riportato in Figura 2.10, il modello di Norman si suddivide in 7 stadi:

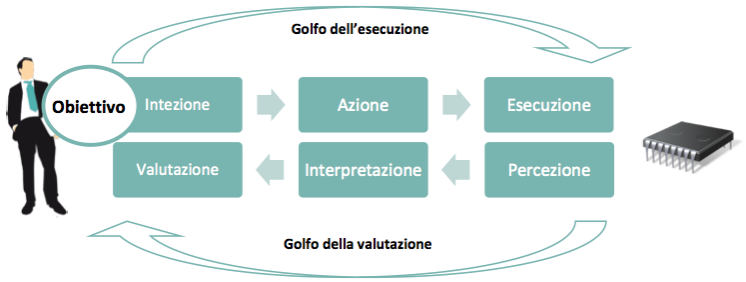

Figura 2.10: Il ciclo d'interazione di Norman.

- **Formulazione dell'obiettivo:** specifica ad alto livello di ciò che si vuole realizzare;
- **Formulazione delle intenzioni:** specifica più dettagliata di ciò che serve per raggiungere l'obiettivo;
- **Specifica delle azioni:** indicazioni di quali azioni sono necessarie per realizzare le intenzioni;
- **Esecuzione delle azioni:** selezione dei comandi necessari per compiere le azioni;
- **Percezione dello stato del sistema:** fase in cui l'utente valuta quanto è stato effettivamente compiuto, confrontando le azioni specificate e l'esecuzione avvenuta;
- **Interpretazione dello stato del sistema:** fase in cui l'utente utilizza la propria conoscenza del sistema per interpretare quanto eseguito dal sistema stesso;
- **Valutazione dei risultati:** fase in cui l'utente confronta lo stato del sistema, in base a quanto egli ha percepito ed interpretato, con le intenzioni, decidendo di conseguenza quale sarà la prossima azione.

Secondo *Norman*, il ciclo d'interazione ha inizio con la formulazione di un *obiettivo*, a cui segue lo stadio di *esecuzione* ed infine quello di *valutazione*. Queste ultime due fasi rappresentano una misura della distanza cognitiva, ovvero della quantità e qualità delle informazioni da elaborare da parte dell'utente per colmare quelli che sono stati chiamati, rispettivamente *golfo dell'esecuzione* (differenza tra le intenzioni dell'utente e le azioni consentite dal sistema) e *golfo della valutazione* (differenza tra output del sistema e quello che l'utente si aspetta) [19].

Il modello di Norman, che ben si adatta all'interazione uomo-computer (HCI), presenta però delle lacune nel campo dell'interazione uomo-robot (HRI), caratterizzato da sistemi complessi, a controllo dinamico, autonomi e cognitivi, operanti in ambienti reali e, di conseguenza, mutevoli. A tal proposito, Scholtz [30] evidenzia la necessità, nel campo dell'interazione uomo-robot, di evidenziare diversi tipi di interazione in base a:

- Natura fisica dei robot;
- Numero di sistemi con cui l'utente è chiamato ad interagire simultaneamente;
- Ambiente in cui avviene l'interazione.

A seguito di tale osservazione, egli definisce una serie di tipologie di interazione, proponendo, per ciascuna di esse, un modello appropriato. in particolare Scholtz individua la seguente classificazione:

- **Supervisor Interaction:** l'interazione supervisionata è caratterizzata dalla necessità di monitorare e controllare che il robot raggiunga il proprio obiettivo. Nella Figura 2.11 è proposto il modello per questo tipo di interazione: il ciclo principale è rappresentato dal loop percezione/valutazione, in quanto le azioni sono, in questo caso, generate automaticamente dal software del robot. Questa modalità d'interazione è fortemente percettiva e necessita di essere supportata sia dal livello delle azioni che da quello delle intenzioni.

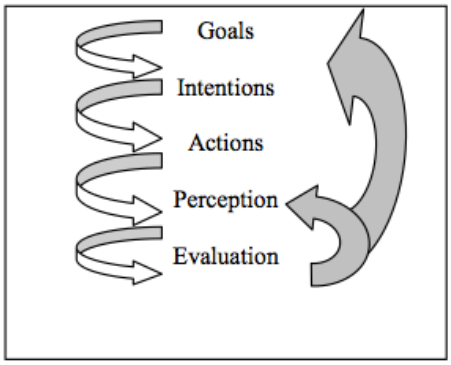

Figura 2.11: Modello per la Supervision Interaction.

- **Operator Interaction:** l'interazione di tipo operatore richiede l'intervento dell'utente per modificare il software interno o il modello su cui si basa il robot, quando il comportamento di quest'ultimo non è accettabile. Come risulta evidente dal modello riportato in Figura 2.12, questa tipologia d'interazione si basa sul controllo della correttezza delle azioni compiute e sulla conformità di queste rispetto all'obiettivo a lungo termine prefissato.

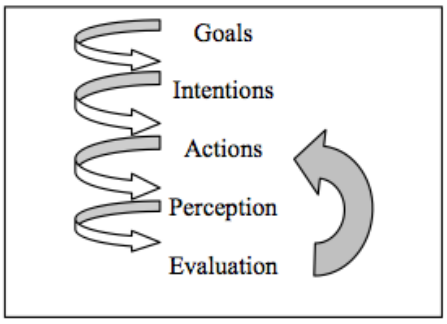

Figura 2.12: Modello per la Operator Interaction.

- **Mechanic Interaction:** questo tipo d'interazione si basa sull'intervento fisico dell'essere umano sulle parti meccaniche del robot. Risulta indispensabile, tuttavia, anche in questo caso, verificare che le modifiche apportate siano coerenti con il comportamento desiderato. Proprio per tale ragione, il modello fornito per questa modalità (Figura 2.13) non si discosta di molto da quello proposto per la *Operation Interaction*. L'unica differenza è data dal fatto che, benché le modifiche siano state applicate dal punto di vista hardware, i test sul comportamento ottenuto debbano essere comunque svolte via software e, di conse-

guenza, entrambi gli aspetti rivestono in questa tipologia d'interazione particolare importanza per ottenere dal robot il comportamento desiderato.

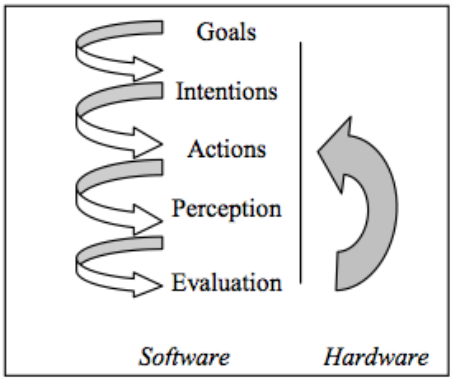

Figura 2.13: Modello per la Mechanic Interaction.

- **Peer Interaction:** l'interazione alla pari richiede un livello di comportamenti superiore rispetto a quelli permessi dall'Operator Interaction, così come i partecipanti all'interazione comunicano tra di loro in termini più elevati di intenzione. Espressioni come *Seguimi, Aspetta finché...* diventano formule di dialogo ragionevole tra uomo e robot in un'interazione alla pari. In queste condizioni, l'osservazione diretta e l'input percettivo sono gli aspetti preponderanti per la fase di valutazione (Figura 2.14). Nel caso in cui il comportamento realizzato non sia corretto, il collaboratore umano ha la possibilità di tramutare la Peer Interaction in una Operator Interaction, intervenendo così alla correzione delle problematiche riscontrate.

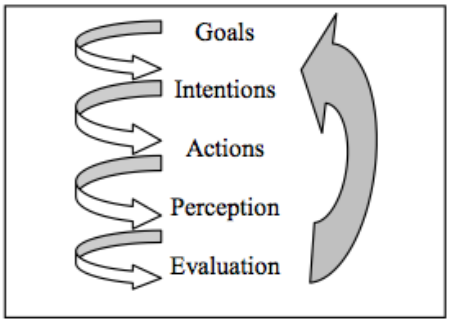

Figura 2.14: Modello per la Peer Interaction.

- **Bystander Role:** questa modalità d'interazione fa riferimento a tutti i casi in cui l'essere umano non opera direttamente con o sul robot, bensì svolge solo un ruolo da spettatore. Anche in queste condizioni, infatti, si può individuare un sottoinsieme di azioni d'interesse: lo spettatore, ad esempio, può far si che il robot smetta di camminare quando di trovano faccia a faccia. L'utente, dunque, in questo caso ha solo un sottoinsieme di azioni disponibili e non è in grado di agire direttamente sul livello dell'obiettivo o delle intenzioni (Figura 2.15). La grande innovazione di questa modalità d'interazione riguarda la procedura con cui informare lo spettatore delle capacità del robot che ha, in quel preciso momento, sotto il suo controllo. Proprio da qui nasce la ricerca sulle emozioni e sull'interazione sociale con i robot.

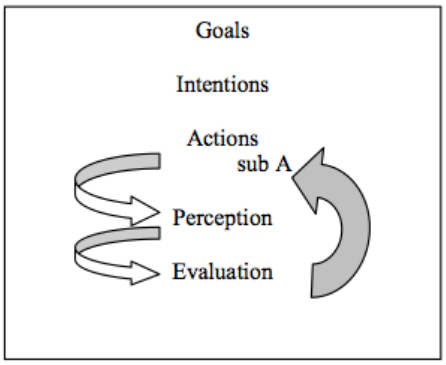

Figura 2.15: Modello per la Bystander Role.

2.2.2 Le modalità d'interazione

Focalizzando l'attenzione sull'interazione sociale tra uomo e robot, numerose ricerche hanno evidenziato l'importanza, in tal contesto, dell'utilizzo di modalità interattive naturali (*human-friendly*), tipiche dell'interazione uomo-uomo, in grado di coinvolgere tutti i sensi umani e i canali di comunicazione come il parlato, la visione, la gestualità ed il tatto.

Capelli e Giovannetti [8], in particolare, hanno individuato sei categorie principali di modalità d'interazione naturale:

- **Parlato:** l'interazione con un robot per mezzo della voce (sia per dare istruzioni che per ricevere risposte) costituisce uno degli obiettivi fondamentali nello sviluppo di interfacce uomo-robot. Situazione tipica in cui l'uso del parlato risulta vantaggioso e l'interazione con robot mobili di servizio, sopratutto in ambienti domestici, dove il robot è libero

di muoversi e mal si presta a ricevere comandi ed a fornire feedback attraverso dispositivi classici;

- **Gesti:** il riconoscimento dei gesti umani è un'area di ricerca in continuo sviluppo e diversi studi si sono interessati al ruolo dei gesti nell'interazione uomo-robot, proponendo innumerevoli tecniche di riconoscimento e relativa produzione di gesti in fase di conversazione e collaborazione;

- **Espressioni Facciali:** nell'ambito dell'interazione uomo-robot, la capacità di riconoscere e produrre espressioni facciali permette al robot di allargare le proprie capacità comunicative, consentendogli, da un lato, di interpretare le emozioni che si dipingono sul volto del proprio interlocutore, dall'altro, di tradurre i propri intenti comunicativi in espressioni da modellare sulla propria faccia robotica. Un esempio ricorrente in letteratura in merito alla capacità di un robot di produrre espressioni facciali è *Kismet*, il robot costruito presso l'*Artificial Intelligence Laboratory* del MIT (Figura 2.16). Grazie ai 15 gradi di libertà presenti nella sola testa (sopracciglia, orecchie, labbra, palpebre e bocca) ed altri 4 gradi di libertà di cui è dotata la piattaforma su cui è posizionato, Kismet è in grado di riprodurre una vasta gamma di espressioni facciali;

Figura 2.16: Alcune possibili espressioni del robot Kismet.

- **Tracciamento dello sguardo:** la direzione dello sguardo gioca un ruolo importante nell'interazione sociale umana e, in particolare, nell'identificazione del focus di attenzione di una persona. Un sistema robotico dotato di questa funzionalità, in grado cioè di identificare dove una persona sta guardando ed a cosa sta prestando attenzione, sarà capace di capire, ad esempio, se questa persona si sta rivolgendo al robot o ad un altro essere umano;

- **Prossemica e cinesica:** modalità di comunicazione più sofisticate sono la *prossemica* e la *cinesica*: la prima concerne la distanza tra gli interlocutori, la variazione della quale può fornire un utile indizio circa la disponibilità o la reticenza alla conversazione. Esempi di prossemica sono stati condotti ricorrendo al sopracitato Kismet: la sua testa, infatti, reagisce alla distanza del proprio interlocutore e quando questo si avvicina troppo invadendo il suo spazio personale, il robot, come mostrato nella Figura 2.17, si ritrae per segnalare il proprio disagio. La cinesica, invece, è la modalità che riguarda il movimento e l'assunzione di posizioni: si occupa dei gesti compiuti utilizzando una o più parti del corpo ed in particolare l'uso delle mani, della mimica facciale e della postura. Questa gestualità, se opportunamente interpretata, risulta essere una preziosa fonte aggiuntiva di informazione;

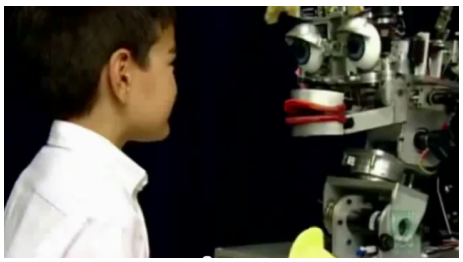

Figura 2.17: Kismet che si ritrae all'avvicinarsi del bambino, risultato delle sue analisi di prossemica.

- **Aptica:** tutto ciò che concerne il senso del tatto può essere classificato sotto la voce di *aptica*, che si può definire come lo studio dell'acquisizione dell'informazione e della manipolazione attraverso il tatto. La realizzazione di interfacce aptiche sofisticate, utilizzabili con robot sociali, è strettamente legata alla conoscenza dell'aptica umana; la comprensione delle abilità percettive, motorie e cognitive dell'utente, infatti, sono indispensabili per la realizzazione di un'interfaccia aptica uomo-robot. Un esempio dei risultati ottenuti in quest'area di ricerca (oltre alla foca *Paro*, presentata nel paragrafo precedente) è *Leonardo*

(Figura 2.18). Questo piccolo robot, infatti, è stato rivestito da una soffice pelle sintetica capace di percepire e localizzare la pressione; la densità dei sensori sparsi sul corpo del robot varia in funzione della frequenza con la quale una certa area entra in contatto con gli oggetti e le persone (maggiore sulle mani e minore sulla schiena).

Figura 2.18: Il robot Leonardo.

Come appare evidente dagli studi mostrati, nella maggior parte dei casi l'interfaccia utilizzata per comunicare con un sistema robotico è ottenuta combinando varie modalità d'interazione, ricorrendo quindi ad una *interfaccia multimodale*. Perchè l'interazione uomo-robot abbia successo, infatti, è necessario sfruttare le tecniche a disposizione per far si che il robot manifesti il più possibile la sua intenzionalità, ossia l'uomo deve poter credere che il robot abbia opinioni, desideri e intenzioni [13].

2.2.3 Il concetto di *Engagement*

Numerosi studi hanno evidenziato come la costruzione di un'interazione efficiente ed efficace sia fortemente legata al concetto di *engagement*, ossia di *coinvolgimento*. La nozione di *coinvolgimento* è stata definita da Sidner e dai suoi collaboratori [14] come quel processo grazie al quale due o più persone stabiliscono, mantengono e terminano il loro stato di connessione durante l'interazione intrapresa.

Nel contesto di una conversazione, oratore ed ascoltatore sono partecipanti attivi e sincronizzati: l'oratore comunica il proprio scopo ricorrendo al linguaggio verbale e non, l'ascoltatore riceve queste informazioni ed è compito dell'oratore comprendere, percepire ed interpretare i suoi comportamenti. In particolare, affinché una conversazione abbia successo, l'ascoltatore deve ricorrere alle sue risorse di attenzione, percezione ed intelligenza per comprendere ciò che l'oratore sta cercando di comunicargli; in altre parole, non

c'è comunicazione se non v'è coinvolgimento del destinatario [12].

Gli studi svolti da Sidner ed altri [14] [12] hanno dimostrato come il coinvolgimento sia un elemento centrale non solo per l'interazione uomo-uomo, ma anche per l'interazione uomo-robot. In particolare, si è osservato come movimenti tipici della conversazione *human-like*, quali sono i movimenti della testa e degli occhi, influiscano notevolmente anche sulle interazioni tra essere umano e robot.
Rivestono particolare importanza, in tal contesto, i robot *Mel* e *Robovie* (Figura 2.19), costruiti il primo presso il MERL, il secondo mediante la collaborazione tra due università giapponesi (Future University - Hakodate e Saitama University).

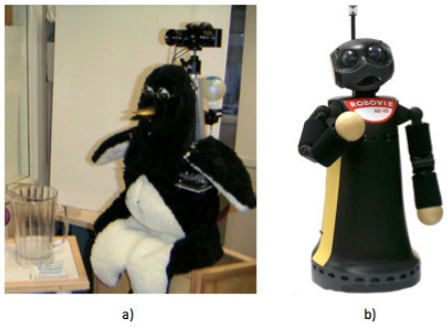

Figura 2.19: a) Il pinguino Mel costruito presso il MERL b) Robovie, il robot guida per un museo, costruito in Giappone.

Entrambi i progetti hanno permesso di evidenziare come i movimenti della testa del robot, eseguiti in momenti chiave della conversazione, incrementino il livello di coinvolgimento dell'interlocutore. Il robot Mel, ad esempio, è in grado di interagire con i propri interlocutori parlando, gesticolando e seguendo con lo sguardo il proprio ascoltatore; esso, inoltre, è in grado di interpretare i gesti del visitatore e determinare il suo grado di interesse all'interazione (Figura 2.20).

Figura 2.20: Il pinguino Mel durante un'interazione.

Dagli esperimenti condotti da Sidner [14] è risultato evidente che se il linguaggio verbale permette un incremento nell'attenzione dell'interlocutore, la sincronizzazione di quest'ultimo con il linguaggio non verbale incrementa notevolmente il grado di coinvolgimento dell'interazione.

Analisi simile a quella eseguita su Mel, è stata sperimentata anche su *Robovie*: un robot avente lo scopo di svolgere il ruolo di guida in un museo, in grado di muovere la testa in momenti significativi dell'interazione con il visitatore (Figura 2.21).

Figura 2.21: Il robot Robovie durante l'esposizione di un quadro ad un visitatore dell'Ohama Museum of Art in Giappone.

I creatori di Robovie [37] hanno ritenuto significativo svolgere due tipologie di esperimenti:

- **Modalità sistematica:** modalità in cui il robot rivolge la sua testa alternativamente all'utente ed all'oggetto che sta spiegando. compiendo tali movimenti in punti significativi del discorso;

- **Modalità non-sistematica:** modalità in cui il robot muove la testa in punti non significativi della conversazione.

Dai grafici riportati in Figura 2.22 risulta evidente come l'utente manifesti attivamente quanto recepito nella modalità in cui il robot muove la testa a punti significativi della comunicazione (si osservi infatti la percentuale di *nodding* (annuire) nel caso sistematico, rispetto a quello non-sistematico; in tale modalità, tuttavia, non si è riscontrato un aumento significativo di scambio di sguardi e ciò può essere dovuto al fatto che l'utente è portato ad imitare i movimenti del robot. Partendo dal presupposto che i casi in cui l'annuire o lo scambio di sguardi tra uomo e robot avvengano contemporaneamente siano indicativi di un elevato livello di coinvolgimento ed

attenzione dell'interlocutore, risulta evidente dal grafico riportato in Figura 2.22.b quanto un'interazione in modalità sistematica sia decisamente più coinvolgente ed efficace.

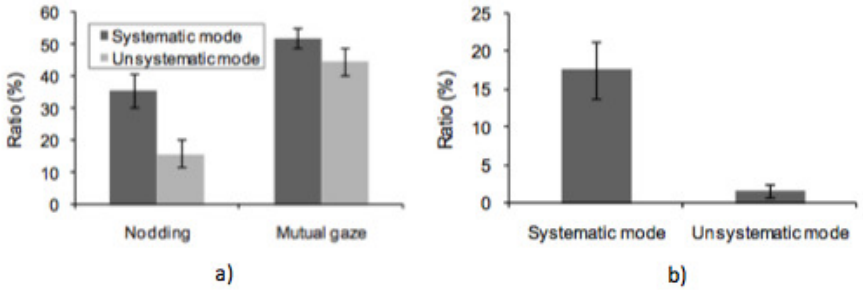

Figura 2.22: a) Livello di partecipazione (valutato in base al movimento della testa) nelle due modalità proposte. b) Percentuale di casi in cui l'annuire o lo scambio di sguardi tra robot e uomo avvengono contemporaneamente.

A fronte degli studi riportati, si può dunque affermare che per ottenere un'interazione uomo-robot di successo sia indispensabile che il robot stesso manifesti il proprio coinvolgimento e che riesca a percepire il livello d'interesse del suo interlocutore, reagendo di conseguenza. Perché ciò sia possibile, è necessario che comportamenti verbali e non verbali siano pienamente integrati, autonomi e progettati perché riproducano le principali caratteristiche dell'interazione uomo-uomo.

2.3 Il linguaggio del corpo: le espressioni del volto umano

Nel paragrafo precedente è stato evidenziato come, per ottenere un'interazione efficace ed efficiente, sia di fondamentale importanza per un robot percepire il livello di coinvolgimento del suo interlocutore. Non esiste tuttavia coinvolgimento senza un adeguato livello di attenzione e tale informazione può essere fornita direttamente dell'interlocutore, mediante il suo volto, i suoi gesti, la sua postura. I comportamenti e le espressioni di una persona, infatti, possono rappresentare una notevole fonte d'informazione.

Un pioniere in questo settore è sicuramente lo psicologo statunitense *Paul Ekman*, il quale ha elaborato il *sistema di codifica delle espressioni facciali (FACS)*: si tratta di uno standard comune per classificare sistematicamente l'espressione fisica dalle emozioni [6], basandosi sull'identificazione dell'attività di rilassamento o contrazione di uno o più muscoli facciali, denominate *action units (AUs)* alcune delle quali sono riportate in Figura 2.23.

Figura 2.23: Alcuni esempi del lavoro svolto da Ekman

Il lavoro di Ekman si è rilevato molto utile non solo nell'ambito della psicologia e dell'animazione, ma anche come linea guida per la creazione di sistemi di riconoscimento delle espressioni facciali. Uno sviluppo rilevante, in tal senso, è stato dato dalle ricerche di El Kaliouby e Robinson [28], i quali, basandosi sulle AUs definite all'interno del sistema FACS ed andando a considerare un'immagine frontale del viso in diverse posizioni della testa, hanno sviluppato un sistema in grado di riconoscere e classificare le espressioni del volto umano in real-time.

Il sistema proposto si basa su due step principali: il primo passo consta nell'identificazione dei movimenti della testa e del volto mediante l'utilizzo di particolari punti di riferimento, riportati nella Figura 2.24. I punti individuati vengono poi quantizzati, combinati temporalmente in sequenza e vengono identificate le sequenze significative mediante un *modello di Markov nascosto (Hidden Markov Model - HMM).*

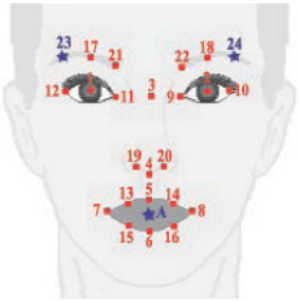

Figura 2.24: Punti di riferimento utilizzati per estrarre i movimenti della testa e del volto

Il secondo passo, invece, consiste nel combinare le sequenze significative individuate mediante una rete bayesiana dinamica (*Dynamic Bayesian Nertwork-DBN*), in modo da generare in uscita l'indicazione dell'espressione riscontrata.

Per comprendere al meglio il funzionamento di questo meccanismo, in Figura 2.25, è stata riportata la struttura complessiva del sistema: come si può osservare, su ogni frame sono stati individuati i punti di riferimento per la testa e per il volto e, mediante questi, si è risaliti alle relative AUs. L'applicazione del classificatore HMM, in seguito, ha permesso di generare un modello temporaneo di quanto osservato ed infine, mediante l'utilizzo del DBN si è riusciti ad individuare lo stato d'animo dell'interlocutore.

Questa tipologia di studi, unita alle ricerche sul tracciamento dello sguardo (presentate nella sezione *Modalità d'interazione)* ed alle indagini sula posizione dei movimenti della testa, hanno permesso di ottenere risultati significativi circa la definizione del livello d'attenzione di un interlocutore coinvolto in un'interazione uomo-robot.

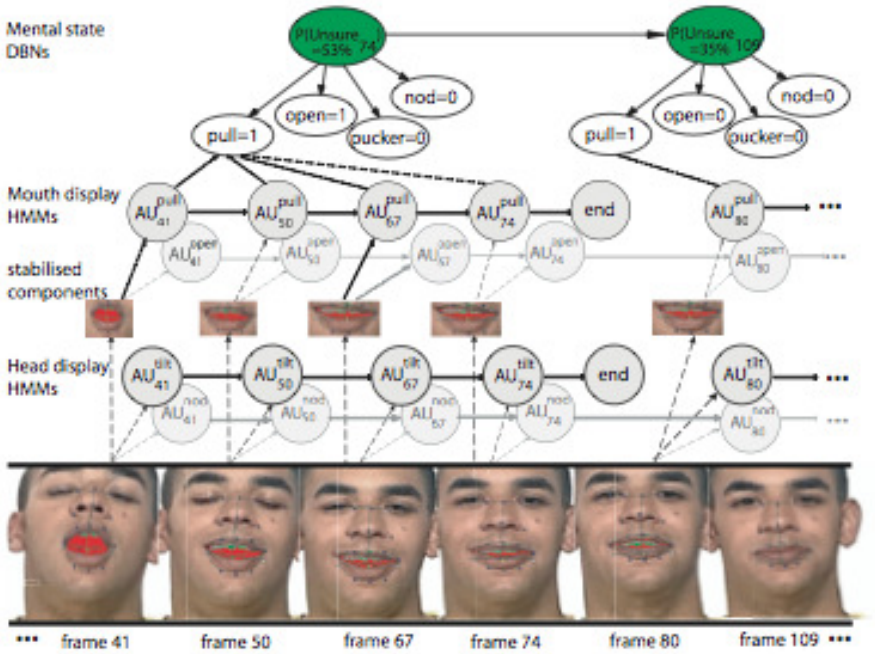

Figura 2.25: Struttura completa del meccanismo di percezione dello stato d'animo.

Capitolo 3

Tecnologie Hardware e Software

"Una cosa fatta, può essere fatta meglio..."

Gianni Agnelli

Il presente elaborato prende in considerazione il robot E-2? (Figura 3.1.a), sviluppato e costruito all'interno del laboratorio di intelligenza artificiale e robotica (*AIRLab*) del Politecnico di Milano.

E-2? si compone di tre parti principali: testa, collo e base; la testa del robot (Figura 3.1.b), oltre a contenere al suo interno tutta la meccanica ed i servomotori per il controllo di occhi, sopracciglia e bocca, sostiene il nuovo sensore Kinect, utilizzato per il sistema di visione artificiale. Il collo (Figura 3.1.d) è stato recentemente riprogettato per garantire, da un lato, una migliore rigidità di tutta la struttura e, dall'altro, per aumentare l'insieme dei movimenti riproducibili. Esso è costituito da 5 servomotori che garantiscono i gradi di libertà necessari per compiere anche movimenti particolarmente complessi. La base del robot (Figura 3.1.c), infine, contiene al suo interno due motori elettrici per il controllo delle ruote che ne permettono il movimento, un PC-brick e cinque batterie da 12 V (quattro per l'alimentazione del PC-brick e delle parti meccaniche ed una per il Kinect).

In questo capitolo illustreremo tutte le tecnologie utilizzate per la realizzazione di questo progetto, con particolare attenzione alle componenti hardware e software del robot.

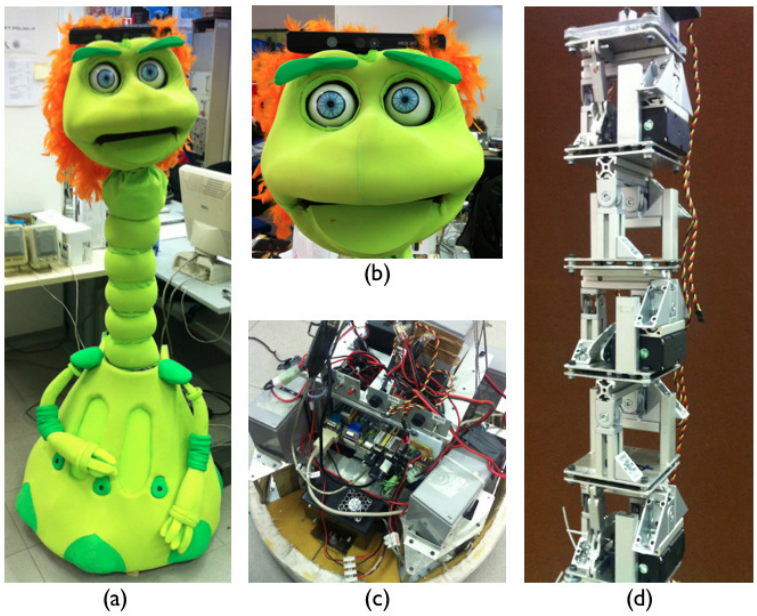

Figura 3.1: Il robot E-2? (a) ed i suoi componenti: testa (b), base (c) e collo (d).

3.1 Componenti Hardware

In questa sezione andremo a descrivere i principali componenti hardware del robot E-2? soffermandoci, nello specifico, sui componenti che hanno avuto un ruolo fondamentale per lo sviluppo del presente elaborato.

3.1.1 Calcolatore

Il cuore del robot E-2? è costituito da un Personal Computer PCBrick con scheda madre Mini-ITX equipaggiata con un processore Intel Core2Duo T7200@2GHz, 2GB di memoria RAM e con un Hard Disk da 80GByte per l'installazione del sistema operativo (Linux Ubuntu 11.04).

Tale piattaforma riesce a garantire una sufficiente potenza computazionale per gli scopi del presente elaborato, senza sacrificare eccessivamente spazio e consumi.

3.1.2 Sonar

I dispositivi sonar forniscono un semplice metodo per ottenere la misura della distanza tra un oggetto ed il trasmettitore stesso. La cintura di sonar di cui è dotato il robot, sviluppata ed assemblata in AIRLab, è realizzata con

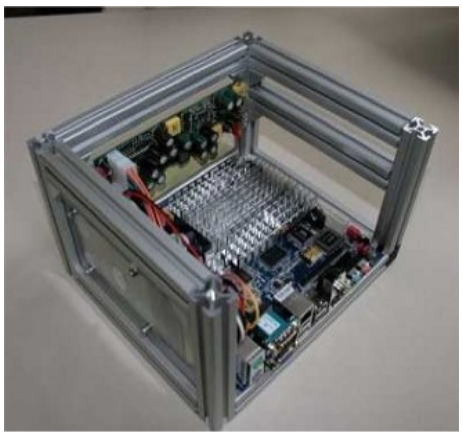

Figura 3.2: PCBrick robot E-2?.

7 diversi sensori Maxbotix EZ-2, collegati ad un unico microcontrollore che si occupa di sequenziare i sensori e trasmettere al PC le distanze misurate. I dispositivi utilizzati, in particolare, sono in grado di rilevare la presenza di oggetti o persone ad una distanza compresa tra 0 e 6 metri.

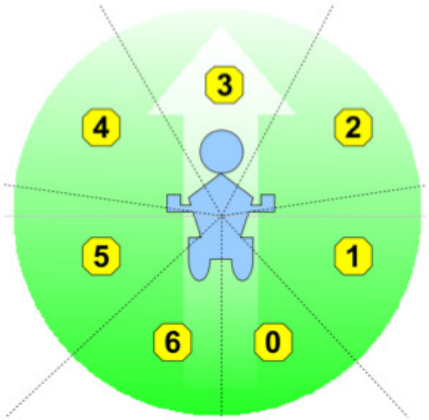

Figura 3.3: Posizione dei sonar sulla base del robot E-2?.

Per ottenere una completa copertura della zona operativa del robot i sensori sono stati disposti come mostrato nella Figura 3.3 permettendo così la rilevazione di un qualsiasi ostacolo su tutta l'area circostante al robot. In particolare:

- **Sensori 0,1 e 2:** permettono la rilevazione di un ostacolo nell'area frontale del robot (circa 120°);

- **Sensori 3,4,5 e 6:** permettono la copertura della restante area laterale e posteriore del robot (restanti 240°);

3.1.3 Motori

Il movimento del robot è attuato per mezzo di due motori elettrici posti nella parte inferiore e controllati da apposita scheda con microcontrollore, il quale riceve i comandi dal calcolatore di bordo (PC-brick) tramite interfaccia seriale. Due ruote libere poste frontalmente e posteriormente, invece, garantiscono stabilità ed equilibro della struttura.

3.1.4 Servomotori

Una delle principali novità introdotte in questa versione del robot E-2? è quella costituita dall'introduzione di una scheda di controllo per la gestione di tutti i servomotori della testa e del collo.

Il servo-controller utilizzato, mostrato in Figura 3.4, fa parte della serie Mini Maestros ed in particolare si tratta della seconda versione di schede di controller Pololu. Le schede Mini Maestro sono disponibili in tre versioni, capaci di comandare rispettivamente 12, 18 e 24 servomotori contemporaneamente. Quest'ultima è quella che è stata scelta per il robot E-2?.

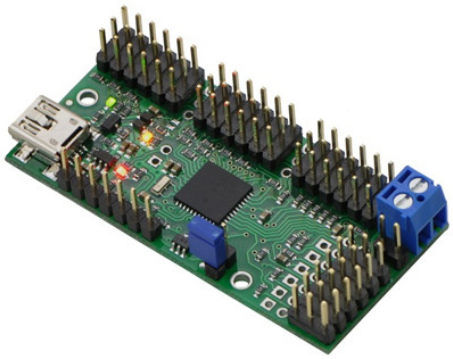

Figura 3.4: Servo-controller Mini Maestros utilizzata sul robot E-2?.

Una delle principali caratteristiche della scheda di controllo utilizzata è sicuramente la possibilità di interfacciamento tramite tre protocolli:

- **USB:** per la connessione diretta con il calcolatore;

- **TTL (seriale):** per l'utilizzo all'interno di sistemi embedded;
- **Script interni:** gestiti direttamente dal controller;

I 24 canali possono essere inoltre configurati per essere utilizzati con sistemi RC (Radio Control Servos) o ESCs (Electronic Speed Control), come uscite digitali o come ingressi digitali ed analogici. I controlli di velocità e accelerazione implementati su ogni canale della scheda permettono, inoltre, di ricreare movimenti dolci e graduali che ben si adattano ad applicazioni di robotica ed animatronica.

La scheda in esame è stata utilizzata per il controllo dei 5 servomotori che costituiscono la struttura del nuovo collo di E-2?, per il controllo dei quattro servomotori di occhi e sopraciglia, per il controllo del movimento della bocca e per la gestione dei movimenti della testa.

3.1.5 Altoparlanti

Nella versione precedente del robot, gli altoparlanti erano stati posizionati all'interno della testa del robot stesso. Tuttavia, per motivi di spazio, eccessivo peso (vista anche l'aggiunta del Kinect) ed una cattiva resa, gli altoparlanti sono stati posizionati sulla parte anteriore del corpo di E-2?.

Figura 3.5: Speaker utilizzati sul robot E-2?.

3.1.6 Kinect

Fino ad oggi, il robot E-2? era equipaggiato con una micro camera UVC (USB Video Class) comunemente utilizzata come camera incorporata in numerosi notebook. Le dimensioni ridotte dell'ottica (che ne favoriva, infatti, il facile occultamento) e la risoluzione di 640x480 pixel offriva il giusto compromesso tra dimensioni e performance per gli scopi preposti. Per lo

sviluppo del presente elaborato, tuttavia, la camera è stata sostituita con un *Kinect* che è stato posizionato sopra la testa del robot come mostrato nella Figura 3.6.

Figura 3.6: Kinect posizionato sopra la testa del robot E-2?.

Originariamente conosciuto con il nome in codice di Progetto Natal [7], il Kinect è un dispositivo per il rilevamento dei movimenti progettato e commercializzato da Microsoft come periferica aggiuntiva per la console Xbox360, al fine di permettere all'utente di interagire con essa senza l'utilizzo di alcun controller ma tramite gesti e comandi vocali.

La motivazione di tale scelta è da ricercarsi nelle potenzialità di tale dispositivo di motion sensing. Il Kinect, infatti (prodotto commercializzato ad un prezzo poco inferiore ai 100 euro), mette a disposizione numerosi sensori spesso utilizzati nell'ambito della visione artificiale quali una camera RGB, un sensore di profondità, una camera IR, un array di microfoni, un accelerometro ed un piede motorizzato che ne permette il movimento. Il posizionamento invece è stato dettato dalla necessità di visualizzare nel modo migliore il volto dell'utente con cui il robot si troverà ad interagire facilitando, quindi, l'analisi dei comportamenti e delle espressioni dell'interlocutore. Tale posizionamento è stato stimato anche sulla base dell'angolo di veduta del dispositivo, che consta di circa 57° orizzontalmente e di 43° rispetto alla verticale; il piede motorizzato, inoltre, consente di inclinare il dispositivo di un angolo (+30°, -30°) rispetto all'asse orizzontale.

Il corpo del sensore, contenuto in una barra orizzontale, è connesso con un piccolo piedistallo motorizzato ideato per essere posizionato sopra o sotto il televisore utilizzato con la console. Il corpo del dispositivo contiene una camera RGB, un sensore di profondità, un accelerometro ed un array di microfoni. Il controllo del dispositivo avviene tramite firmware proprietario che consente riconoscimento ed analisi del movimento di un corpo in una scena 3D ed il riconoscimento dei comandi vocali impartiti dall'utente.

Sia il sensore di profondità che la camera RGB, integrati all'interno del dispositivo, producono un segnale a 30 Hz (30fps) con una risoluzione di 640x480 pixel; 8-bit VGA nel caso della camera RGB, 11-bit VGA nel caso del sensore di profondità che permette di discernere tra 2,048 livelli di sensitività. Tale sensore, nello specifico, è costituito da un proiettore laser infrarosso combinato con un sensore CMOS monocromatico, il quale permette di catturare dati video in 3D sotto qualsiasi condizione di luce [36] [35].

L'array di microfoni [27] formato da 4 microfoni, processa un segnale audio a 16-bit @16KHz per ogni canale. Il dispositivo è infine alimentato tramite connessione USB supportata con un'alimentazione da 12V aggiuntiva la quale ha richiesto l'inserimento di un'ulteriore batteria all'interno del robot.

La principale innovazione proposta dal Kinect [17] [34] [20] consiste nel software proprietario incluso al suo interno che consente un avanzato riconoscimento di gesti e movimenti. Il software permette, infatti, il riconoscimento contemporaneo all'interno della scena di un massimo di 6 utenti per due dei quali è possibile l'estrazione di 20 giunti tramite skeleton capability. Tale funzionalità permette, previa sincronizzazione tramite l'assunzione da parte dell'utente di una determinata posa per alcuni secondi, l'estrazione dello schema dello scheletro dell'utente e l'identificazione dei 20 giunti principali.

Nello scorso febbraio, Microsoft ha annunciato l'imminente rilascio del Kinect Software Development Kit (SDK) per ambiente Windows Seven, avvenuto poi nel corso del mese di Luglio 2011 [26] [24]. L'SDK rilasciata include al suo interno i driver per l'utilizzo del dispositivo su un qualsiasi sistema con Windows Seven installato, dando quindi la possibilità di sviluppare applizazioni C, C o Visual Basic sfruttando le seguenti informazioni provenienti direttamente dai sensori del dispositivo:

- **Raw Sensor Streams:** accesso agli stream di basso livello del sensore di profondità, della camera RGB ed ai microfoni;

- **Skeletal tracking:** accesso alla funzionalità che permette di tracciare l'immagine delle parti principali dello scheletro umano di una o due persone inquadrate dal dispositivo;

- **Funzionalità audio avanzate:** Audio processing. Sono incluse numerose funzioni di rimozione del rumore, di identificazione della sorgente del suono ed integrazione con le API di riconoscimento vocale di Microsoft Windows;

Per quanto riguarda il mondo openSource, invece, si fa riferimento ad OpenKinect. Una community nata con lo scopo di creare un SDK che avrebbe permesso a tutti gli utenti Microsoft e non di sviluppare applicazioni sfruttando le potenzialità dei dispositivi di interazione quali il Kinect. I driver utilizzati per l'accesso ai sensori del Kinect sono i libfreenect [22], scaricabili in modo completamente gratuito da internet e compatibili con OS X, Linux e Windows.

Successivamente, l'organizzazione no-profit OpenNI in partnership con PrimeSense, formata con lo scopo di promuovere la compatibilità e l'interoperabilità di dispositivi, applicazioni e middleware di interazione naturale, ha rilasciato gratuitamente l'OpenNI Framework, che mette a disposizione un gran numero di API per la scrittura di applicazioni utilizzanti l'interazione naturale. Le API attualmente a disposizione coprono sia una comunicazione tramite dispositivi di basso livello (come sensori audio e video) sia con sistemi di alto livello e soluzioni middleware (computer vision e tracking). Tra i dispositivi supportati vi è anche il Microsoft Kinect da noi utilizzato.

Anche ROS (Robot Operating System), l'ambiente di sviluppo da noi utilizzato per il controllo del robot E-2? e di cui si parlerà nel dattaglio nel corso del capitolo, ha seguito gli sviluppi delle comunità openSource mettendo a disposizione opportuni moduli per accedere ai dati sensoriali del Kinect. Il rilascio dell'ultima versione ROS, detta Electric, ha sancito il passaggio al framework OpenNI sul quale ci siamo basati per l'acquisizione dei dati necessari allo sviluppo dei moduli di visione.

Il modulo ROS, nominato Openni, ha messo a nostra disposizione gli stream video delle camere RGB, IR e Depth del sensore ed ha fornito l'accesso alle API sviluppate nel framework OpenNi.

3.2 Componente Software

In questa sezione andremo a descrivere ROS (Robot Operating System), ambiente utilizzato per lo sviluppo del sistema di controllo del robot che, nel contesto di questo elaborato, ha sostituito il precedente Framework MRT (Modular Robot Tolkit) [2], sviluppato dal Politecnico di Milano.

3.2.1 ROS: Robot Operating System

ROS [5] è un meta-sistema operativo progettato e sviluppato per la gestione ed il controllo di robot. Esso mette a disposizione servizi tipici dei comuni sistemi operativi, quali astrazioni hardware, controllo di dispositivi di basso livello, implementazione delle funzionalità più utilizzate, scambio di messaggi tra processi ed un sistema di gestione dei propri pacchetti. A questi si devono aggiungere tool e librerie per scrittura, compilazione ed esecuzione di codice su più computer.

ROS è, dal punto di vista concettuale, suddivisibile in tre diversi livelli [4]:

- **Filesystem level**: definisce uno standard per la gestione e per la creazione dei moduli che comporranno il sistema di controllo;
- **Computation Graph level**: l'albero di computazione è gestito come una rete peer-to-peer di processi o nodi in esecuzione utilizzanti un'apposita infrastruttura di comunicazione;
- **Community level**: strumenti messi a disposizione per la pubblicazione ed il mantenimento dei pacchetti all'interno della community;

Nel linguaggio ROS si indica con il termine *Package* un'unità software, la quale può essere rappresentata da un processo ROS detto *nodo*, da una libreria ROS-dependent, da dataset, da file di configurazione o da qualunque altro insieme di entità abbia senso aggregare.
All'atto della creazione, un package è corredato da un file manifest.xml, il quale si occupa di definire i metadati che caratterizzano l'elemento. In questo caso i metadati possono indicare informazioni circa la licenza del pacchetto, le dipendenze, i flags per il compilatore ed altre informazioni che verranno utilizzate dal sistema per l'utilizzo del pacchetto stesso.

Più package che mettono a disposizione funzionalità aggregabili possono essere, inoltre, organizzati a loro volta all'interno di uno stack. Anche in questo caso ogni stack è corredato dal proprio stack.xml che, come nel caso dei package, si occuperà di definirne i metadati caratteristici.

Il Computation Graph è una rete peer-to-peer di processi ROS attualmente in esecuzione. Le entità di cui è costituita tale rete sono principalmente le seguenti:

- **Nodi**: i nodi sono processi che svolgono una computazione. ROS è stato progettato per essere un sistema altamente modulare e, pertanto, ogni modulo dovrebbe occuparsi di un ristretto numero di obiettivi come il comando delle ruote del robot, la gestione di un sensore come un sonar o un laser etc. Per la definizione di un nodo vengono tipicamente utilizzate due librerie messe a disposizione del sistema: *roscpp* e *rospy* rispettivamente per i linguaggo C++ e Python;

- **Master**: il Master mette a disposizione un nameserver e relativi servizi di lookup a tutti i nodi facenti parte del computation graph. Tale servizio permette quindi ai vari nodi di scambiarsi messaggi o sfruttare determinati servizi messi a loro disposizione;

- **Message**: l'architettura di comunicazione implementata in ROS utilizza un sistema di messaggi per la comunicazione tra nodi. Sono supportati tutti i principali tipi primitivi (integer, float, boolean, etc) ed array degli stessi e, con essi, possono essere integrate varie tipologie di dati semplici o composti, similmente a quanto avviene nelle struct per il linguaggio C;

- **Topic**: lo scambio di messaggi è regolato tramite un paradigma di publish and subscribe. Un nodo, quindi, invia un messaggio pubblicandolo su un determinato topic che identifica il contenuto del messaggio pubblicato. Un nodo che risulta interessato ad una certa tipologia di dati dovrà quindi registrarsi all'apposito topic. Ci possono essere nodi concorrenti che pubblicano o si sottoscrivono allo stesso topic come è possibile che un solo nodo possa pubblicare e sottoscriversi a diversi topic. In generale publisher e subscriber non sono a conoscenza della loro esistenza in quanto l'idea di base del paradigma è quella di disaccoppiare la produzione di dati ed informazioni dal loro consumo;

- **Services**: il paradigma publish and subscribe mette a disposizione dell'utilizzatore un'elevata flessibilità, ma è un sistema di comunicazione

molti-a-molti e non si adatta alle interazioni di tipo request/reply. Per questi scopi ROS mette a disposizione metodi per la creazione di appositi servizi. I servizi sono creati tramite la definizione di due particolari messaggi destinati alla gestione della richiesta e della rispettiva risposta. Un nodo quindi, potrà pubblicare un determinato servizio che potrà essere sfruttato da un secondo nodo, il quale invierà la richiesta e si metterà in attesa della risposta;

- **Bag**: i bag sono un utile ed importante strumento per la memorizzazione dei dati contenuti all'interno dei messaggi.

L'architettura appena descritta permette, quindi, di sviluppare un sistema di controllo anche molto complesso attraverso la creazione di semplici nodi che utilizzino l'architettura di comunicazione per lo scambio di dati ed informazioni.

3.2.2 L'architettura software

Come spiegato nel paragrafo precedente, l'utilizzo di ROS ha imposto una suddivisione del sistema di controllo delle varie parti del robot in singole entità o nodi. Alcune componenti hanno, inoltre, richiesto di essere trasportate dall'ambiente MRT, per il quale erano state progettate, all'ambiente RO, andando così a ridefinire il sistema di gestione della comunicazione (unica vera differenza tra i due sistemi).

L'architettura modulare dell'intero sistema di controllo è riportata schematicamente nella Figura 3.8 all'interno della quale si possono identificare i principali stack di visione, di interazione e di controllo.

Per quanto riguarda lo stack di visione, chiamato *vision-modules*, esso contiene tutti i moduli di visione implementati al fine di estrarre le feature di interesse a partire dallo streaming video del Kinect. Le analisi condotte sono state suddivise all'interno di due modul che agiscono in cascata.

- **user-tracker**: il riconoscimento di un utente all'interno della scena avviene tramite l'utilizzo delle librerie OpenNI, le quali permettono di accedere direttamente ai dati sensoriali del Kinect. Questo modulo ha quindi il compito di individuare l'utente migliore attualmente sulla scena e pubblicare frame by frame all'interno del topic *com* la posizione del relativo centro di massa in coordinate x, y e z;

- **head-analyzer**: in questo modulo sono incapsulati vari algoritmi che permettono di estrapolare le informazioni relative all'interlocutore con cui il robot sta interagendo. Algoritmi di *face detection*, *eyes detection*, *optical flow* e di *classificazione* permettono nello specifico l'estrazione di tutte le principali feature necessarie per condurre le successive analisi. Tutti i dati raccolti vengono, frame dopo frame, comunicati al resto del sistema tramite la pubblicazione di appositi messaggi sui topic *headDataMSG, eyesDataMSG, movesDataMSG*.

Entrambi i precedenti moduli fanno riferimento alle informazioni sensoriali messe a disposizione dai driver OpenNI contenuti nello stack principale di ROS.

Il secondo stack utilizzato all'interno del presente progetto prende il nome di *hw-modules* e contiene tutti i moduli necessari per la comunicazione con i componenti hardware del robot; in particolare:

- i motori delle ruote del robot;
- il motore del Kinect;
- la cintura di sonar.

L'architettura a messaggi utilizzara in ROS non permette nativamente la creazione di un flusso informativo bidirezionale che possa essere sfruttato da un lato per l'invio dei comandi da catturare e, dall'altro, per fornire utili informazioni di stato e di feedback indispensabili per l'implementazione di un controllo robusto. A tale scopo si è scelto di utilizzare le librerie *actionlib* (appositamente incluse in ROS) che permettono la creazione di un sistema che possa sfruttare un paradigma client-server come quello mostrato nella Figura 3.7 [1].

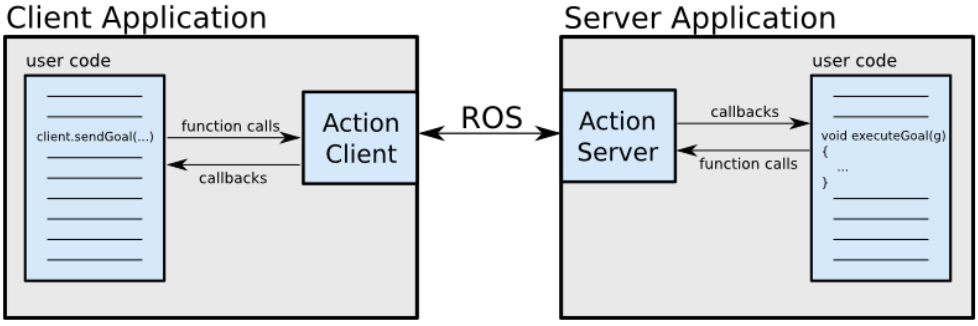

Figura 3.7: Architettura messa a disposizione dalle librerie actionlib.

Tale meccanismo è stato quindi sfruttato per la creazione di singoli server che permettessero di attuare particolari comandi e, nel contempo, che potessero in automatico fornire informazioni circa l'attuale stato di funzionamento, notificando così la conclusione di un task. Tale sistema è stato quindi utilizzato per attuare i comandi dei motori delle ruote del robot e del motore del Kinect ed in particolare è stato applicato nei seguenti due moduli:

- **kinect-motor**: si tratta di un modulo di tipo *server* che mette a disposizione opportune primitive per il controllo incrementale della posizione del sensore e, pertanto, agevola la fase di analisi condotta nel modulo *head-analyzer*;

- **wheel-motor**: anche in questo caso si tratta di un modulo server avente l'obiettivo di attuare i comandi impartiti ai motori del robot e rendere disponibile, istante dopo istante, lo stato di esercizio degli stessi.

All'interno dello stack *hw-modules*, oltre ai precedenti due moduli è presente anche il modulo *sonar* avente il compito di rendere facilmente accessibili tutti i dati provenienti dalla cintura di sonar montata a bordo del robot.

Lo stack *interaction-modules*, invece, si occuperà di gestire l'attuazione dei comportamenti del robot andando ad implementare:

- il controllo dei servomotori del collo della testa e delle sue componenti;

- il controllo di un sintetizzatore vocale che permetta di riprodurre le frasi pre-impostate in opportuni file di configurazione.

Lo stack in esame contiene al suo interno i seguenti moduli ROS, tutti implementati tramite l'utilizzo delle librerie actionlib.

- **speak-api**: l'architettura client-server messa a disposizione dalle librerie actionlib è risultata particolarmente adatta per la realizzazione di un modulo di sintesi vocale che permetta di riprodurre frasi pre-impostate;

- **head-api**: tramite le librerie actionlib, si è implementato un modulo avente l'obiettivo di riprodurre determinate espressioni del volto del robot mediante l'attuazione di opportuni script atti al controllo dei servomotori di occhi, sopracciglia e bocca;

- **neck-api**: tale modulo consente di modificare in modo incrementale la posizione del collo rispetto alla posizione di riposo (verticale). All'interno del modulo, inoltre, sono presenti anche alcune primitive per la riproduzione di movimenti complessi come, ad esempio, quello dell'inchino.

I tre moduli sopra indicati permettono di gestire parallelamente i controlli che essi mettono a disposizione, occupandosi inoltre automaticamente di gestire azioni complesse fornendo metodi per la verifica dello stato di attuazione del comportamento richiesto e funzioni di feedback.

Il controllo vero e proprio, invece, avviene all'interno del modulo *e2-brain* che si occuperà di gestire l'gli algoritmi utilizzati per il controllo del movimento del robot e del processo d'interazione. In particolare:

- **Controllo di movimento**: è stato utilizzato Mr.Brain [2], sistema fuzzy precedentemente utilizzato all'interno di MRT per il quale si è reso necessario un porting all'interno dell'ambiente ROS. Il meccanismo di controllo, sulla base dei dati sensoriali provenienti dalla cintura di sonar e sulla base delle informazioni provenienti dall'algoritmo di user detection, consente di ottenere in uscita i set-point di velocità tangenziale e rotazionale da fornire ai motori;

- **Processo di interazione**: sulla base dei dati estratti dai moduli di visione, è stata modellata una macchina a stati che permette di seguire e controllare tutti gli stati di evoluzione dell'interazione. Sulla base dei dati a disposizione, verranno quindi determinati i set-point dei servomotori del collo e della testa e verranno indicate le frasi da riprodurre tramite l'utilizzo del sintetizzatore vocale.

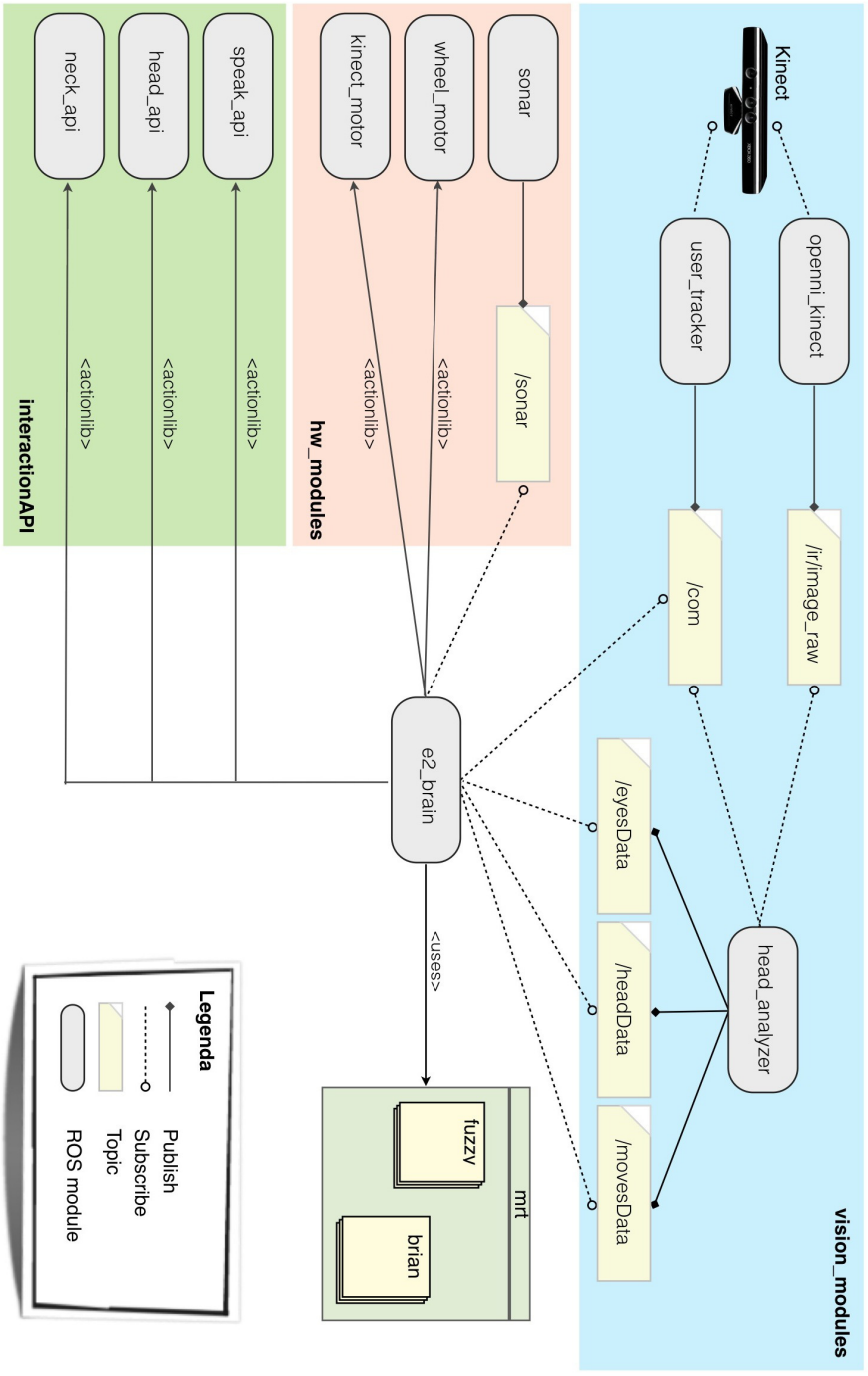

Figura 3.8: Suddivisione in moduli del software di controllo del robot.

Capitolo 4

Il sistema di visione

"Morpheus: Che vuol dire reale? Dammi una definizione di reale. Se ti riferisci a quello che percepiamo, a quello che possiamo odorare, toccare e vedere, quel reale sono semplici segnali elettrici interpretati dal cervello. Questo è il mondo che tu conosci. Il mondo com'era alla fine del XX secolo. E che ora esiste solo in quanto parte di una neuro-simulazione interattiva che noi chiamiamo Matrix. Sei vissuto in un mondo fittizio, Neo."

Matrix

Uno degli obiettivi primari della Computer Vision è l'identificazione di oggetti e corpi all'interno di immagini digitali bidimensionali o 3D. Il riconoscimento di oggetti in tempo reale è uno dei requisiti fondamentali per numerose applicazioni robotiche ed, anche in questo caso, lo sviluppo del modulo di visione è stato un passo fondamentale per lo sviluppo del presente elaborato.

Il sistema di visione da noi implementato si pone come principale obiettivo quello di fornire una serie di informazioni di base (posizione dell'utente, riconoscimento del volto, riconoscimento degli occhi e dei movimenti) che possano essere sfruttate per la determinazione del livello di interesse e di attenzione dell'interlocutore.

La maggior parte dei sistemi esistenti, come evidenziato nel Capitolo 2, basano il proprio funzionamento sull'identificazione di alcune particolari attività muscolari del volto umano (dette action units -AUs) e sull'utilizzo del *Sistema di Codifica delle Espressioni Facciali (FACS)* sviluppato da Paul Ekman [18]; tali applicativi, tuttavia, ricorrono ad immagini ad elevata definizione, oltre che ad ambienti operativi controllati; caratteristiche man-

canti per il caso di studio da noi esaminato.

A fronte di ciò, come suggerito dagli studi svolti da El Kaliouby e Robinson [29], si è optato per l'individuazione dei ben più riconoscibili movimenti del volto (occhi, bocca sopracciglia, etc.), oltre che dei movimenti della testa lungo i suoi tre assi principali.

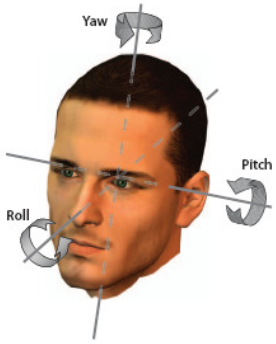

Figura 4.1: Rappresentazione dei gradi di libertà della testa dell'uomo.

Grazie alla classificazione delle possibili espressioni facciali fornita dal *Robotics Institute* dell'università di *Camegie Mellon*, unita agli studi sul linguaggio del corpo presenti in letteratura, siamo stati in grado di costruire un sistema per l'individuazione del grado di attenzione dell'interlocutore, che sia valido anche in presenza di immagini a medio-bassa risoluzione e di ambienti operativi particolarmente rumorosi.

Oltre alle problematiche di riconoscimento dell'utente, del volto e delle feature di nostro interesse, uno dei principali ostacoli che si è dovuto affrontare, nello sviluppo del sistema qui presentato, è stato quello relativo ad una forte limitazione imposta dal firmware del Kinect: i Driver ROS utilizzati, infatti, non hanno permesso lo stream simultaneo dei flussi video RGB ed IR. Tale mancanza, probabilmente, è da ricercarsi nel meccanismo di creazione dell'immagine IR stessa. Per la produzione di tale immagine, infatti, viene utilizzata come base un'immagine in scala di grigi (prodotta dal sensore RGB) sulla quale vengono applicati i dati provenienti dal sensore IR. L'immagine ottenuta da tale sovrapposizione è quella che viene restituita dai Driver sotto il nome di immagine IR.

Tale limite ha imposto un forte vincolo sugli algoritmi da utilizzare per

l'analisi delle immagini a disposizione. Se, infatti, l'utilizzo della camera RGB è ottimo per un'analisi generale della scena e conseguente riconoscimento del volto dell'interlocutore, la camera IR permette di semplificare notevolmente il riconoscimento degli occhi ed, in particolar modo, di eliminare gran parte degli elementi di background, senza l'utilizzo di algoritmi dedicati. La differenza tra le due immagini è ben evidente all'interno della Figura 4.2.

Figura 4.2: Confronto tra un'immagine in scala di grigi (sx) e IR (dx).

Una soluzione a tale limite si sarebbe ottenuta con il posizionamento di una seconda camera, in modo da poter accedere ad entrambi gli stream in modo simultaneo. Tale operazione, tuttavia, avrebbe reso vano l'intento di ricorrere ad un unico dispositivo di facile gestione (come è il Kinect) per la parte sensoristica di visione, senza dimenticare le problematiche circa il posizionamento della nuova camera e la sua sincronizzazione con lo stream generato dal Kinect.

Si è dunque scelto di sfruttare le informazioni del solo stream IR che ha permesso, in generale, di semplificare notevolmente le analisi svolte dal modulo di visione, soprattutto per quanto riguarda la reiezione del rumore. Tale scelta, tuttavia, ha nel contempo ridotto la capacità di analisi del robot impedendo, ad esempio, il riconoscimento e l'analisi della bocca dell'utente. Nonostante ciò, i risultati ottenuti dimostrano come le feature estratte siano comunque sufficienti per gli scopi del presente elaborato ed, inoltre, la semplificazione imposta ha permesso l'ottenimento di un'analisi real-time.

4.1 Architettura del sistema di visione

Il sistema di visione sviluppato nel presente elaborato è costituito da tre diversi moduli, aventi l'obiettivo di fornire tutte le informazioni utili al fine

di eseguire una stima del grado di attenzione e di interesse dell'interlocutore. Lo stack ROS *vision-modules*, come mostrato nella Figura 4.3, è costituito da tre differenti moduli.

- **openni-kinect**: modulo ROS che mette a disposizione le interfacce del framework OpenNI e pubblica, in accordo con la sintassi ROS, gli stream RGB, IR e Depth provenienti dal Kinect in altrettanti topic, affinchè possano essere comodamente utilizzati all'interno dal sistema;
- **user-tracker**: modulo ROS che permette di rilevare il centro di massa e la stima della posizione della testa dell'interlocutore con cui condurre l'interazione. Tali dati verranno comunicati al resto del sistema tramite pubblicazione all'interno del topic *com*;
- **head-analyzer**: modulo ROS che, sulla base dei dati provenienti dallo *user-tracker* e sfruttando lo stream IR dato dal modulo *openni-kinect*, ha l'obiettivo di estrarre informazioni utili alla stima del grado di attenzione e di interesse dell'interlocutore.

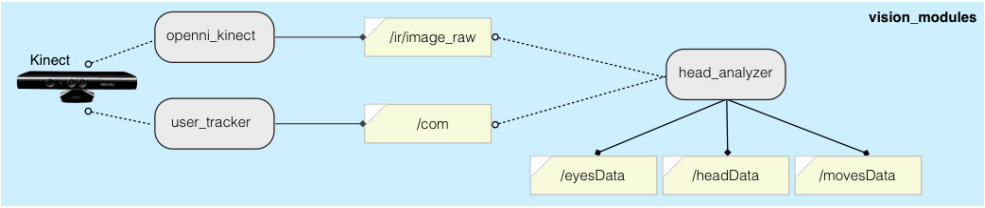

Figura 4.3: Architettura del sistema di visione.

4.1.1 Il modulo openni-kinect

Il framework OpenNI mette a disposizione un insieme di API per lo sviluppo di applicazioni secondo il principio di *Write Once, deploy eveywhere,* disaccoppiando di fatto le dipendenze tra sensori e middleware [3].

Il modulo ROS *openni-kinect* da noi utilizzato, oltre che garantire l'accesso a tale framework, mette a disposizione un rapido punto di accesso ai dati sensoriali del kinect, sfruttando a pieno l'architettura di comunicazione *message oriented* imposta da ROS.

L'architettura del framework OpenNI, suddivisibile dal punto di vista concettuale su tre livelli, è riportata nella Figura 4.4 dove si possono distinguere:

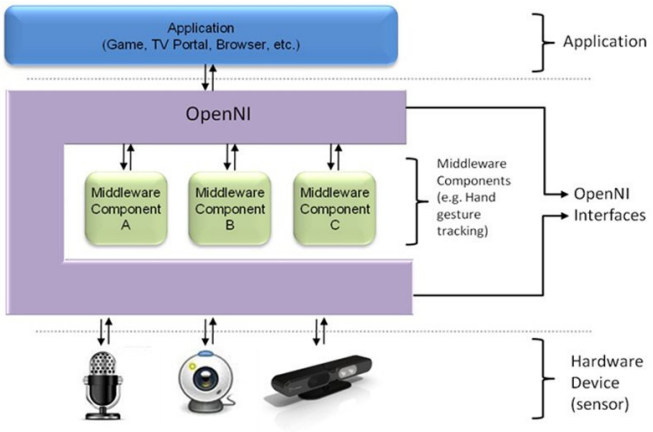

Figura 4.4: Composizione concettuale a tre livelli del framework OpenNI.

- **Livello superiore**: rappresenta una generica applicazione di interazione naturale, costruita al di sopra del framework OpenNI;
- **Livello intermedio**: identifica il framework OpenNI, il quale mette a disposizione interfacce di comunicazione tra i sensori utilizzati ed i componenti middleware che analizzeranno i dati da essi prodotti;
- **Livello inferiore**: racchiude i sensori e dispositivi hardware utilizzati per la registrazione video ed audio della scena da analizzare.

Il framework OpenNI è quindi definibile come un layer astratto che mette a disposizione interfacce sia per dispositivi fisici di basso livello che per componenti middleware più complesse. Tali interfacce consentiranno la registrazione di diversi componenti, che prenderanno il nome di *moduli* e verranno utilizzati per la produzione e l'analisi dei dati sensoriali. I moduli che, allo stato attuale, risultano supportati sono i seguenti:

- **Moduli Sensori**
  - *Sensori 3D*;
  - *Camera RGB*;
  - *Camera IR*;
  - *Device audio*: microfoni o array di microfoni;
- **Componenti middleware**

- *Full body analysis middleware*: componente software per l'analisi di dati sensoriali destinati alla generazione delle informazioni relative ad orientamento, punti di joint, centro di massa dell'intero corpo dell'utente;
- *Hand point analysis middleware*: componente software destinata all'analisi di dati sensoriali per il calcolo della posizione di un hand point;
- *Gesture detection middleware*: componente software atto all'identificazione di movimenti predefiniti come, ad esempio, il movimento di una mano;
- *Scene Analyzer middleware*: componente software per l'analisi della scena, avente l'obiettivo di produrre informazioni circa elementi di background, elementi significativi, identificazione di figure etc.

I dati 3D possono essere definiti come significativi se permettono di capire, comprendere e tradurre la scena corrente. La creazione di tali dati ha inizio con la pubblicazione, da parte di un sensore, dei cosiddetti raw data. Tipicamente tali dati sono rappresentati attraverso una depth map, all'interno della quale ogni pixel è rappresentato sulla base della distanza dal sensore.

All'interno del framework OpenNI, un middleware dedicato viene utilizzato per l'analisi di questi dati, con l'obiettivo di ottenere informazioni di più alto livello e che, quindi, possano essere capite ed utilizzate all'interno delle applicazioni. In questo senso, all'interno di OpenNI sono definiti i cosiddetti *Production note*: un insieme di componenti aventi un ruolo fondamentale nell'elaborazione dei dati richiesti da applicazioni di interazione naturale. I Production node, dunque, sono elementi fondamentali all'interno delle interfacce OpenNI ed hanno il compito di generare specifiche tipologie di dati.

Per chiarire quanto sopra descritto, si prenda in considerazione il seguente esempio, che non si discosta di molto da quanto è stato implementato nel corrente elaborato. *Un'applicazione ha l'obiettivo di tracciare il movimento di una figura umana all'interno di una scena 3D. Tali obiettivi richiedono un nodo di produzione che metta a disposizione i dati del corpo della persona all'applicazione. Tale nodo è noto come user-generator. A sua volta lo user-generator farà uso di un altro nodo di produzione per ottenere i dati necessari alle proprie analisi: il depth-generator. Quest'ultimo nodo, nello specifico, fa uso di un sensore dal quale riceve i cosiddetti raw data per ge-*

nerare la depth map richiesta.

Tra i Production node attualmente supportati vi sono:

- **Device:** nodo che rappresenta un dispositivo fisico come un sensore di profondità o una camera RGB;
- **Depth Generator:** nodo avente l'obiettivo di generare una depth map. Tale sistema sarà implementato tramite l'ausilio di un qualsiasi sensore 3D che possa essere utilizzato con OpenNI;
- **Image Generator:** nodo avente l'obiettivo di generare una image-map a colori. Tale sistema sarà implementato con l'ausilio di un qualsiasi sensore a colori che possa essere utilizzato con OpenNI;
- **IR Generator:** nodo avente l'obiettivo di generare una mappa IR. Tale sistema sarà implementato tramite l'ausilio di un qualsiasi sensore IR che possa essere utilizzato con OpenNI;
- **Audio Generator:** nodo avente l'obiettivo di generare uno stream audio tramite l'utilizzo di un generico dispositivo audio, connesso e compatibile con OpenNI;
- **User Generator:** nodo avente l'obiettivo di generare una rappresentazione completa o parziale del corpo umano all'interno di una scena 3D;

I precedenti nodi, in unione con lo stream IR disponibile sul topic ROS */camera/ir/image-raw*, sono stati utilizzati per l'implementazione dei rimanenti due moduli di visione che si andranno ora a descrivere nel dettaglio.

4.1.2 Il modulo user-tracker

Il modulo *user-tracker* è stato implementato con lo scopo di fornire i dati relativi alla posizione dell'interlocutore all'interno della scena 3D.

Come mostrato nella Figura 4.5, il modulo in esame utilizza le interfacce ed i Production node messi a disposizione dal framework OpenNI, per determinare la presenza di un nuovo possibile interlocutore all'interno della scena fornendone, frame dopo frame, la posizione in coordinate x,y (origine al centro dell'immagine) e z (distanza tra l'utente ed il sensore).
Il modulo è caratterizzato da una prima sezione all'interno della quale vengono eseguite le inizializzazioni ROS ed OpenNI. Nello specifico:

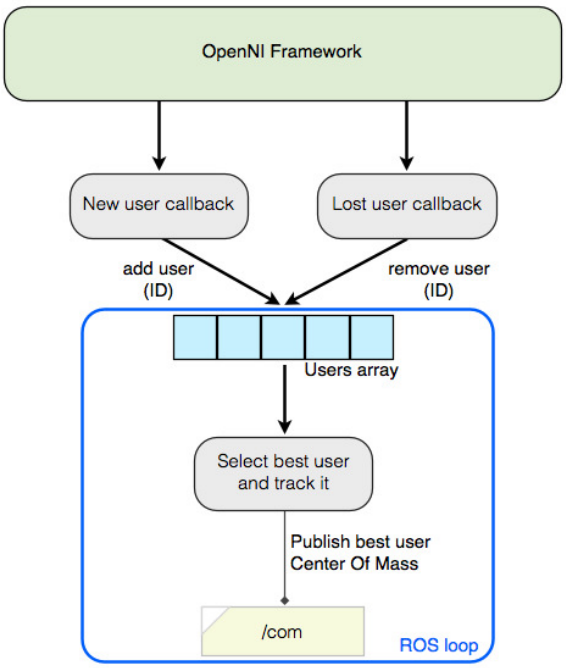

Figura 4.5: Architettura generale del modulo user-tracker.

```
//Ros node initialization
ros::init(argc, argv, "user_tracker");
ros::NodeHandle nh;
```

Il precedente listato permette di eseguire un re-mapping dei parametri eventualmente passati da riga di comando e di specificare il nome del nodo in esame (in questo caso *user-tracker*) che, per convenzione, dovrà essere unico all'interno di tutto il sistema. La seconda riga, invece, permette di creare un handler per il nodo corrente, il quale si occuperà di eseguire le inizializzazioni necessarie per l'esecuzione dello stesso.

Per utilizzare le interfacce messe a disposizione dalle librerie OpenNI, invece, è sufficiente definire un nuovo contesto tramite le seguenti righe di codice, andando a specificare, come file di configurazione, quello fornito all'interno del modulo ROS *openni-kinect*.

```
//OpenNI initialization
string configFilename = ros::package::getPath("
    openni_tracker")+"/openni_tracker.xml";
nRetVal = g_Context.InitFromXmlFile(configFilename.c_str
    ());
```

La necessità di accedere ad alcune proprietà degli utenti rilevati all'interno della scena (quali distanza e centro di massa) ha reso necessario l'utilizzo del Production node *user-generator*, introdotto nel paragrafo precedente e gli utenti rilevati da tale nodo verranno poi inseriti all'interno di un'apposita struttura dati tramite l'utilizzo di due funzioni di callback opportunamente registrate

```
//User generatior production node
nRetVal = g_UserGenerator.Create(g_Context);
//Callbacks
XnCallbackHandle hUserCallbacks;
g_UserGenerator.RegisterUserCallbacks(User_NewUser,
   User_LostUser, NULL, hUserCallbacks);
//Run
nRetVal = g_Context.StartGeneratingAll();
```

In particolare, la funzione *User-NewUser* ha l'obiettivo di aggiungere un nuovo utente rilevato all'interno del vettore di utenti attualmente riconosciuti. Ogni utente all'interno di tale struttura sarà identificato da un particolare ID che ne permetterà l'identificazione e l'accesso alle relative proprietà (centro di massa, distanza e posizione all'interno della scena corrente).

```
//New user detected
XN_CALLBACK_TYPE User_NewUser(xn::UserGenerator&
   generator, XnUserID nId, void* pCookie);
```

Dualmente a quanto visto in precedenza, la funzione *User-LostUser* ha il compito di gestire gli utenti che, per qualche motivo, non sono più all'interno della scena. L'algoritmo di user detection implementato all'interno del Kinect è in grado, inoltre, di mantenere attivo il tracking di un utente anche se quest'ultimo è uscito dall'inquadratura per alcuni secondi. Tale meccanismo, che i test effettuati hanno constatato essere molto robusto, ci ha permesso di mantenere attiva l'interazione anche nei casi in cui gli spostamenti dell'interlocutore o del robot generino una momentanea perdita dell'utente stesso.

```
//Lost user
XN_CALLBACK_TYPE User_LostUser(xn::UserGenerator&
   generator, XnUserID nId, void* pCookie);
```

La restante parte del modulo è infine caratterizzata dal loop ROS di esecuzione:

```
ros::Rate r(30);
while(ros::ok())  //ROS LOOP
{
```

```
    //....
    //....
    ros::spinOnce();
    r.sleep();
}
```

L'ambiente ROS mette a disposizione alcuni metodi per la gestione del loop di esecuzione; nello specifico è possibile definire un rate massimo di esecuzione che, nel caso in esame, è stato imposto pari alla frequenza di aggiornamento delle camere del Kinect (30 Hz). Se l'esecuzione del modulo dovesse tuttavia richiedere un tempo inferiore a quello a disposizione, il metodo *r.sleep()* si occuperà di mettere il processo in stato di sleep fino al momento utile per la prossima iterazione di esecuzione, rilasciando così tutte le risorse del sistema per altre computazioni. L'istruzione *ros::SpinOnce()*, infine, è necessaria per l'attivazione dei meccanismi di comunicazione e per la gestione delle funzioni di callback.

Per gestire il caso in cui più utenti siano identificati all'interno della scena corrente, condizione particolarmente probabile nel contesto in cui il robot è destinato ad operare, è stato implementato un semplice algoritmo di selezione dell'utente migliore; il codice di tale algoritmo è riportato nel listato sottostante.

```
void getBestUser(XnUserID usersArray [MAX_USERS],
   XnUInt16 nUsers, xn::UserGenerator& generator)
{
  //Find the nearest user
  userData temp;
  //Get user's ID
  nearestUserData.userID = -1;

  //Get User's COM
  nearestUserData.com.Z = max_user_distance;
  //generator.GetCoM(usersArray[0], nearestUserData.com);
  for(int k = 0; k < nUsers; k++)  //For each detected
     user
  {
    //if there is nearest user
    generator.GetCoM(usersArray[k],temp.com);
    if((int)nearestUserData.com.Z > (int)temp.com.Z && (
       int)temp.com.Z != 0)
    {
      nearestUserData.userID = (int)usersArray[k];
      nearestUserData.com = temp.com;
```

```
    }
  }
}
```

La precedente funzione *getBestUser()* ha il compito di individuare tra tutti gli utenti attualmente identificati, l'utente più vicino al robot. Una volta identificato il cosiddetto best user, tutti i relativi dati quali ID, posizione e centro di massa vengono salvati all'interno di un'apposita struttura che andrà via via aggiornata in modo da ottenere un tracking dell'utente all'interno della scena.

```
struct userData {
  int userID;
  //User Center of Mass
  XnPoint3D com;
  //Control Variables
  int i;
  bool fullQueue;
  bool userInRange;
};
```

Tali informazioni saranno particolarmente utili per il comportamento *Go-To-User* che sarà presentato nel dettaglio nel capitolo 5.

Per rendere disponibili a tutti i moduli del sistema le informazioni estratte circa l'utente migliore all'interno della scena, al termine di ogni ciclo di esecuzione il modulo si occuperà di pubblicare sul topic *com* i dati relativi al centro di massa dell'utente ed una stima della posizione della testa (basata sulla posizione del centro di massa all'interno dell'immagine corrente).

Per permettere la pubblicazione di un determinato contenuto è necessario definire nella sezione iniziale del modulo un *publisher* che si occuperà di realizzare tale operazione.

```
ros::Publisher pubCoM = nh.advertise<user_tracker::Com>("
    com", 1000);
```

La sintassi del messaggio è invece definita all'interno del file *msg/Com.msg* all'interno del modulo corrente. Nel caso specifico, il messaggio è costituito da due variabili di tipo *Point32* (punti float a 3 dimensioni a 32 bit) che verranno utilizzati rispettivamente per il salvataggio del centro di massa dell'utente (x, y e z) e per la stima della posizione della testa all'interno dell'immagine corrente (x e y).

```
geometry_msgs/Point32 comPoints
geometry_msgs/Point32 headPoint
```

La funzione *compileMessages()*, infine, si occuperà di compilare il messaggio con i dati disponibili all'istante corrente e di permetterne la pubblicazione all'interno del tipic.

```
#include "user_tracker/Com.h"
user_tracker::Com coms;

void compileMessages()
{
  //Compiling COM message
  coms.comPoints.x = (float)nearestUserData.com.X;
  coms.comPoints.y = (float)nearestUserData.com.Y;
  coms.comPoints.z = (float)nearestUserData.com.Z;

  //Compiling head data
  coms.headPoint.x = (float)nearestUserData.headROIpt1.x;
  coms.headPoint.y = (float)nearestUserData.headROIpt1.y;
}
```

4.1.3 Il modulo head-analyzer

Il modulo *head-analyzer* ha l'obiettivo di analizzare l'immagine corrente al fine di estrarre informazioni quali posizione e movimenti del volto, posizione e dimensioni degli occhi dell'utente etc. L'architettura generale del modulo è schematizzata in Figura 4.6 nella quale sono riportati anche i collegamenti con gli altri moduli dello stack di visione analizzati in precedenza, aventi, come sopra illustrato, l'obiettivo di fornire tutti i dati su cui condurre le analisi.

Dal modulo openni-kinect ed in particolare tramite la sottoscrizione al topic *camera/ir/image-raw*, viene caricato il frame corrente dallo stream IR del Kinect, mentre dal topic */com*, prodotto dal modulo user-tracker, viene letta la posizione del centro di massa dell'utente ed una stima della posizione del volto all'interno del frame corrente.

Dalla struttura generale del modulo, inoltre, si evidenzia come l'analisi del volto dell'utente venga suddivisa su due livelli distinti. Una prima analisi, avente come obiettivo l'identificazione del volto dell'utente, viene condotta quando l'interlocutore si trova ad una distanza inferiore al metro dal robot; una seconda, avente l'obiettivo di identificare tutte le altre feature di interesse (quali la posizione degli occhi) viene condotta, invece, quando l'interlocutore risulta essere a non più di 90 cm.
Tale soluzione è stata adottata per ottenere un duplice scopo:

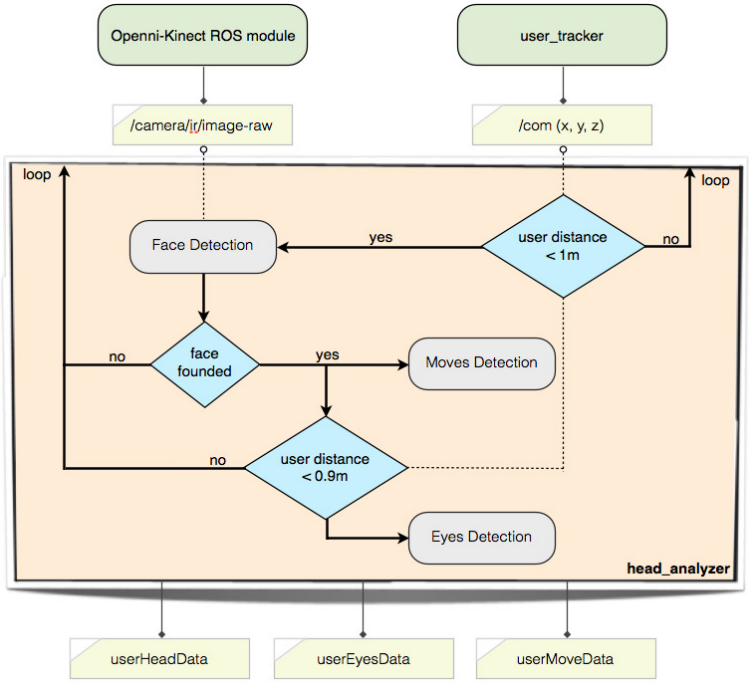

Figura 4.6: Architettura generale del modulo head-analyzer.

Figura 4.7: Variazione della qualità dell'immagine IR con l'aumentare della distanza.

- da una parte per sfruttare al meglio le immagini del sensore IR che degradano di qualità in modo proporzionale con l'aumentare della distanza interlocutore-robot;
- dall'altra per cercare di ottenere un'interazione quanto più reale possibile, cercando di riproporre quelle distanze tipiche dell'interazione uomo-uomo.

Anche in questo caso, la prima parte del modulo è destinata all'inizializzazione dello stesso all'interno del sistema ROS:

```
//Ros initialization
ros::init(argc, argv, "head_analyzer");
ros::NodeHandle nh;
```

Per quanto riguarda il loop ROS di esecuzione, si è anche qui optato per un rate di esecuzione eguale al rate di aggiornamento delle camere del Kinect (30Hz).

```
//ROS LOOP
ros::Rate r(30);
while(ros::ok())
{
  //...
  //...
  ros::spinOnce();
  r.sleep();
}
```

Al fine di ottenere una semplice sincronizzazione tra i vari moduli presenti all'interno dello stack di visione, abbiamo optato per avviare l'esecuzione solamente quando tutti i dati necessari ai suoi scopi sono stati ricevuti; questo è il principale obiettivo del controllo effettuato all'interno del ciclo di esecuzione e di seguito riportato. Si osservi, inoltre, che le analisi avranno inizio solamente quando l'interlocutore sarà stato effettivamente raggiunto dal robot e ad una distanza inferiore al metro.

```
if (IRImageReady && userDistanceReady && getFrame)
{
  IRImageReady = false;
  userDistanceReady = false;
  getFrame = false;

  if(userDistance <= USER_DISTANCE)
  {
    if(detectFace() && detectMovesEnable)
```

```
    {
      if(userDistance <= EYES_TRACKING_DISTANCE)
      detectEyes();
      detectMovements();
    }

    //Display analysis results
    imshow("Head-Analyzer", frameDrawn);

    detectMovesEnable = true; //Enable starting from 2nd
       frame
  }

  frameIR.copyTo(prevFrameIR);
}
```

Per la lettura dei dati computati dagli altri moduli dello stack di visione, in accordo con il paradigma publish and subscribe utilizzato in ROS, è stato necessario iscrivere esplicitamente il modulo in esame ai topic di interesse. L'iscrizione ad un topic avviene tramite:

- la dichiarazione di un *subscriber* per ogni topic di interesse specificando, oltre al nome del topic, una dimensione per il numero di messaggi da mantenere nella coda;
- la funzione di callback, che dovrà essere attivata all'arrivo di un nuovo messaggio all'interno del relativo topic.

```
//Message subscribers
ros::Subscriber subIRKinect = nh.subscribe("camera/ir/
    image_raw", 1000, getKinectIRImage);
ros::Subscriber subUserDistance = nh.subscribe("com",
    1000, getUserData);
```

I valori delle variabili booleane del precedente codice di sincronizzazione verranno, quindi, impostati a valore vero (true) all'interno delle funzioni di callback aventi il compito di accedere ai nuovi dati appena questi sono stati pubblicati sui topic di riferimento.

Fino ad ora è stata descritta solamente la struttura generale del modulo in esame. Nella parte seguente si andrà, invece, ad analizzare nel dettaglio i tre algoritmi che sono stati implementati per il riconoscimento del volto, degli occhi e dei movimenti dell'utente.

L'algoritmo di Face Detection

L'algoritmo di face detection da noi implementato, il cui schema è riportato all'interno della Figura 4.8, è stato sviluppato con lo scopo di sfruttare nel miglior modo possibile l'immagine IR del Kinect. Tale immagine, infatti, è in grado di mettere in forte evidenza il volto dell'interlocutore rispetto ad altri elementi di background che, essendo ad una maggiore distanza dal robot, verranno riprodotti con una tonalità decisamente più scura.

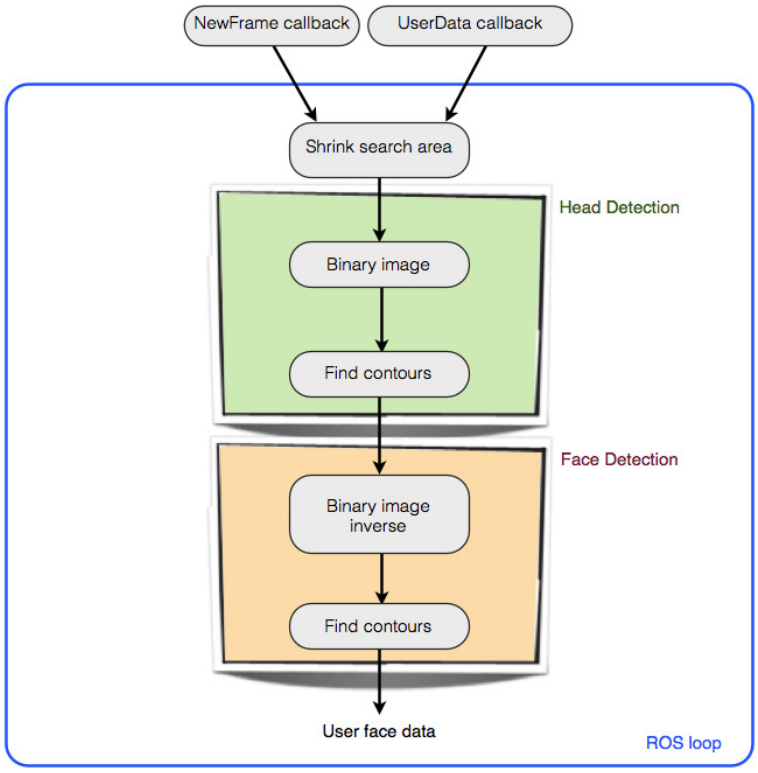

Figura 4.8: Architettura generale dell'algoritmo di face detection.

La stima della posizione del volto dell'utente, effettuata dal modulo user-tracker viene qui utilizzata per eseguire una riduzione significativa dell'area di ricerca del volto. Tale operazione si è resa necessaria sia per ridurre il numero di eventuali falsi positivi riscontrati all'interno dell'immagine, sia per migliorare le prestazioni dell'algoritmo. La riduzione dell'area di ricerca, infatti, diminuisce le dimensioni dell'immagine a cui applicare l'algoritmo da 640×480 pixel dell'originale a circa 200×200 pixel, come mostrato in Figura 4.9.

Figura 4.9: Riduzione dell'immagine originale (sx) in una nuova immagine utilizzata per la ricerca del volto dell'utente (dx).

Una volta applicata la riduzione, l'area di ricerca viene analizzata una prima volta per determinare il rettangolo di interesse (ROI) che contiene al suo interno la testa dell'utente. Questo obiettivo è stato raggiunto applicando il seguente procedimento:

- l'immagine dell'area di ricerca è stata convertita in un'immagine binaria tramite un'opportuna sogliatura determinata empiricamente.
- all'immagine ottenuta si è applicato un algoritmo di ricerca e riconoscimento dei contorni.

Entrambe le operazioni sono state rese possibili tramite l'utilizzo delle librerie OpenCV che, recentemente, sono state importate anche all'interno del sistema ROS.

La creazione dell'immagine binaria si è resa necessaria per ridurre drasticamente il rumore presente all'interno dell'immagine restituita dal sensore IR del Kinect e, quindi, per semplificare notevolmente la ricerca dei contorni. Tale elaborazione è stata possibile applicando una soglia k ad ogni pixel p_{xy}, sulla base della relazione 4.1

$$p_{xy} = \begin{cases} 255, & \text{se } p_{xy} < k \\ 0, & \text{altrimenti} \end{cases} \tag{4.1}$$

In particolare, la soglia k per la creazione dell'immagine binaria è stata calcolata in modo sperimentale, tramite l'esecuzione di diversi test sotto varie condizioni di luce. Al termine della fase di test si è preferito, inoltre, far dipendere il valore di k dalla distanza tra l'interlocutore ed il robot.

```
const unsigned int THRESHOLD_HEAD_NEAR = 220;
const unsigned int THRESHOLD_HEAD_FAR = 150;
```

```
if(userDistance < 750)
{
  //...
  //...
  //Convert IRImage into gray scale image and then into
     binary one
  cvtColor(frameIR, binaryFrame, CV_BGR2GRAY);
  binaryFrame = binaryFrame > THRESHOLD_HEAD_NEAR;
}
else
{
  //..
  //..
  //Convert IRImage into gray image and then into binary
     one
  cvtColor(frameIR, binaryFrame, CV_BGR2GRAY);
  binaryFrame = binaryFrame > THRESHOLD_HEAD_FAR;
}
```

I risultati ottenuti tramite l'applicazione dell'immagine binaria sono riportati all'interno della Figura 4.10.

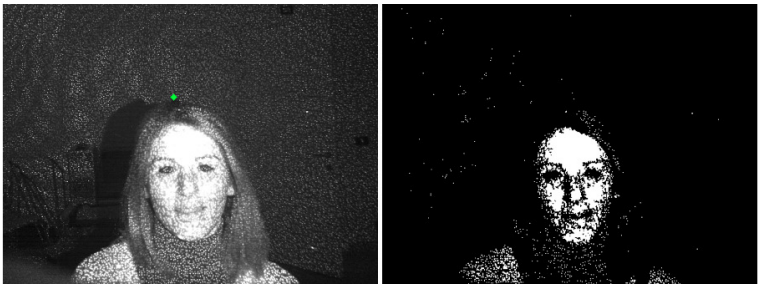

Figura 4.10: Immagine IR (sx) ed immagine binaria (dx) a confronto.

La seconda ed ultima fase dell'algoritmo di head detection proposto consiste nella ricerca dei contorni della testa dell'interlocutore all'interno dell'immagine binaria precedentemente computata. Anche in questo caso sono state utilizzate le librerie OpenCV, le quali mettono a disposizione un'implementazione dell'algoritmo per il riconoscimento dei contorni all'interno di un'immagine (binaria) presentato da Suzuki, S. e Abe, K. nel 1985 [32].

```
//OpenCV find contours algorithm
findContours(headBinaryFrame, contours, hierarchy,
   CV_RETR_CCOMP,
      CV_CHAIN_APPROX_SIMPLE, cvPoint(headROI.x, headROI.
         y));
```

I risultati dell'applicazione della precedente analisi sull'immagine binaria sono riportati all'interno della Figura 4.11, nella quale, per una migliore comprensione, ogni contorno è stato colorato con un colore diverso.

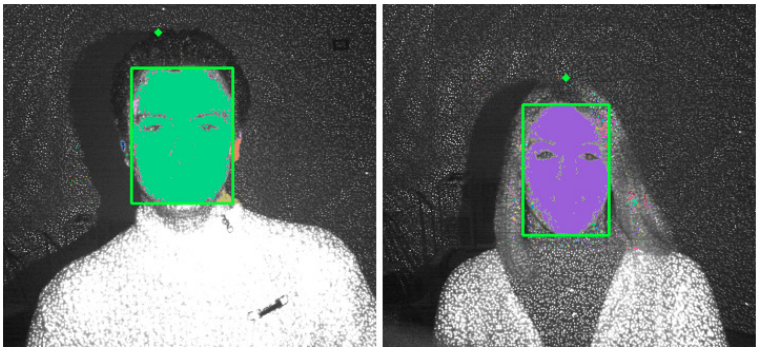

Figura 4.11: Contorni riconosciuti tramite l'utilizzo dell'algoritmo di findcontours implementato nelle librerie OpenCV.

```
//Filter contours and get the biggest one
for (int i = 0; i >= 0; i = hierarchy[i][0])
{
  headROI = boundingRect(contours[i]);

  //Get the biggest area
  int temp = headROI.width * headROI.height;
  if(temp > contourArea)
  {
    contourArea = temp;
    biggerContourIdx = i;
  }
}
```

Il ROI contenente il volto dell'utente, infine, è stato selezionato considerando il contorno più lungo tra quelli restituiti dalla precedente analisi. Tale approssimazione si è dimostrata valida, in quanto l'area di ricerca è stata ridotta sufficientemente da comprendere al suo interno quasi esclusivamente il volto dell'utente.

I dati ottenuti dalle precedenti analisi sono utilizzati come input per l'algoritmo di face detection, il quale ha come obiettivo ridurre ulteriormente le componenti di rumore lungo i bordi del volto, quali capelli ed ombre. Per meglio comprendere lo scopo di questa seconda analisi, nella Figura 4.12 sono riportati i risultati dell'algoritmo di head-detection e dell'algoritmo di face detection.

Figura 4.12: Nelle due immagini riportate, è riportato in verde il ROI ottenuto con l'algoritmo di head detection ed in rosso quello ottenuto con la seconda fase di analisi (face detection).

Dalle immagini risulta evidente come il riconoscimento del volto dell'utente sia più preciso dopo l'applicazione dell'algoritmo di face detection il quale ha permesso, come sopra riportato, di escludere dal ROI della testa componenti di rumore come i capelli. Il raggiungimento di una maggiore precisione ha permesso una semplificazione degli algoritmi di riconoscimento dei movimenti e degli occhi dell'interlocutore.

Anche in questa seconda fase di analisi è stato utilizzato un procedimento molto simile a quello descritto per l'algoritmo di head detection. È stato infatti sfruttato l'algoritmo di ricerca dei contorni, proposto dalle librerie OpenCV, applicandolo, in questo caso, alla porzione di immagine contenuta nel ROI della testa dell'interlocutore.

```
//Find face contours
findContours(headROIframe, contours, hierarchy,
    CV_RETR_CCOMP,
       CV_CHAIN_APPROX_SIMPLE, cvPoint(eyesROI.x, eyesROI.
          y));

//Filter contours and get the biggest one
biggerContourIdx = 0;
contourArea = -1;
for (int i = 0; i >= 0; i = hierarchy[i][0])
{
  faceROI = boundingRect(contours[i]);

  //Get the biggest area
  int temp = faceROI.width * faceROI.height;
  if(temp > contourArea)
```

```
  {
    contourArea = temp;
    biggerContourIdx = i;
  }
}
```

L'algoritmo proposto per il riconoscimento del volto dell'utente ha permesso di ottenere ottimi risultati indipendentemente dalle caratteristiche somatiche dall'interlocutore . L'algoritmo, inoltre, identifica il volto dell'utente indipendentemente dall'inclinazione dello stesso ed indipendentemente dalla posizione assunta (profilo o frontale) come mostrato nella Figura 4.13.

Figura 4.13: Risultati dell'algoritmo di face detection in condizioni particolari.

Si è potuto osservare, infine, come il variare delle condizioni di luce nel corso della giornata, non influiscano in modo sostanziale sui risultati ottenuti. Una volta completata l'analisi, il ROI contenente il volto dell'utente viene nuovamente utilizzato per l'identificazione dei movimenti e per l'identificazione degli occhi.

L'algoritmo di Eye Detection

L'algoritmo di eye detection implementato sfrutta gli stessi principi di funzionamento che sono stati descritti nel paragrafo precedente, in occasione dell'analisi dell'algoritmo di face detection.
Come mostrato schematicamente nella Figura 4.14, una volta riconosciuta la posizione del volto dell'interlocutore all'interno del frame corrente, viene effettuata una seconda analisi avente l'obiettivo di identificare la posizione degli occhi. Per semplificare l'analisi dal punto di vista computazionale, è stata definita una regione di ricerca all'interno del volto dell'utente definita come *EyesArea* (Figura 4.15) le cui dimensioni sono valutate sfruttando le simmetrie e le distanze che ogni volto umano deve rispettare, indipendentemente dalle caratteristiche somatiche dell'individuo.

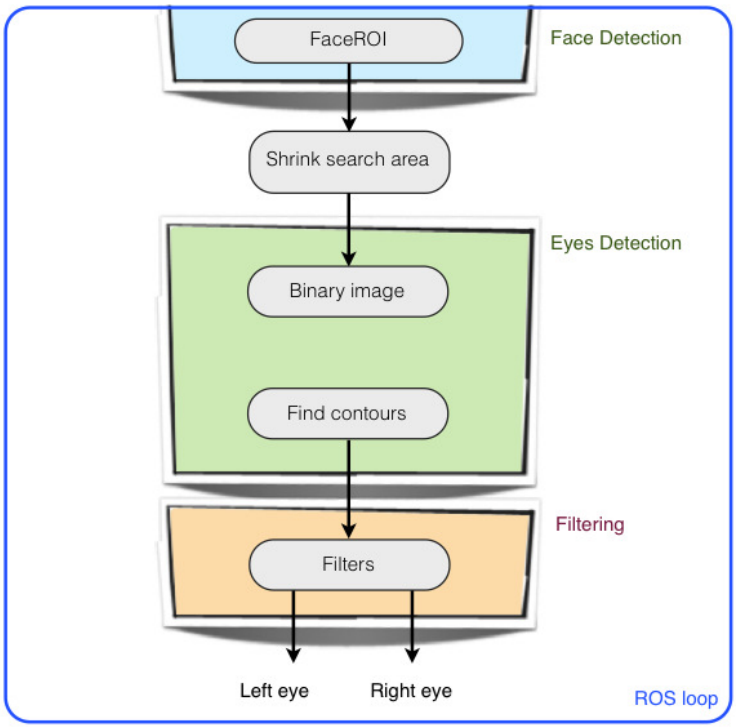

Figura 4.14: Architettura generale dell'algoritmo di eyes detection.

Figura 4.15: Area di ricerca degli occhi (in rosso).

Una volta calcolata la nuova area di ricerca, anche in questo caso è stato necessario applicare alcune trasformazioni aventi il duplice obiettivo di ridurre il rumore e di mettere in evidenza i contorni degli occhi nell'immagine di riferimento. Come fatto in occasione dell'immagine del volto dell'interlocutore, si è optato per la computazione dell'immagine binaria dell'area di ricerca. A differenza del caso precedentemente trattato, tuttavia, è stato scelto di utilizzare una sogliatura adattiva che prendesse in considerazione non solo il valore del pixel p_{xy}, ma anche la media dei valori di tutti gli n pixel nell'intorno del pixel considerato, secondo la relazione 4.2.

$$p_{xy} = \begin{cases} 255, & \text{se } p_{xy} > T(x,y) \\ 0, & \text{altrimenti} \end{cases} \tag{4.2}$$

dove il termine $T(x,y)$ della relazione 4.2 rappresenta la media dei valori di tutti i pixel nell'intorno del pixel p_{xy}. Per ottenere risultati sufficientemente accurati, la dimensione dell'intorno di pixel da considerare per la computazione del valore di $T(x,y)$ è stata dimensionata sperimentalmente sulla base della distanza tra interlocutore e robot.

La seconda transizione, utilizzata per migliorare la fase di riconoscimento dei contorni, è quella di erosione. La funzione di erosione applica una particolare trasformazione all'immagine sorgente per determinare la forma di un insieme di pixel in relazione al quale viene computato il minimo, sulla base della relazione 4.3.

$$dst(x,y) = \min_{(x',y'):element(x',y')\neq 0} src(x+x', y+y') \tag{4.3}$$

La matrice utilizzata per computare la trasformazione di erosione, per motivi computazionali, è stata impostata di dimensione 3×3. Il seguente listato riporta i passi fondamentali compiuti per eseguire le trasformazioni appena descritte

```
//Distance handler
if (userDistance < 700)
{
  //Define blobs dimension
  MIN_EYE_BLOB_SIZE = 30;
  MAX_EYE_BLOB_SIZE = 300;

  //Get binary image and optimize it for blob analysis
  adaptiveThreshold (temp2, temp1, 255,
              ADAPTIVE_THRESH_MEAN_C, THRESH_BINARY_INV,
                 89, 0); //AirLab 125
```

```
  erode(temp1,contoursFrame, Mat());
}
else if ((userDistance >= 700)&&(userDistance < 760))
{
  //Define blobs dimension
  MIN_EYE_BLOB_SIZE = 40;
  MAX_EYE_BLOB_SIZE = 300;

  //Get binary image and optimize it for blob analysis
  adaptiveThreshold (temp2, temp1, 255,
            ADAPTIVE_THRESH_MEAN_C, THRESH_BINARY_INV,
               91, 0); //AirLab 125
  erode(temp1,contoursFrame, Mat());
}
else
{
  //Define blobs dimension
  MIN_EYE_BLOB_SIZE = 35;
  MAX_EYE_BLOB_SIZE = 300;

  //Get binary image and optimize it for blob analysis
  adaptiveThreshold (temp2, temp1, 255,
             ADAPTIVE_THRESH_MEAN_C, THRESH_BINARY_INV,
                75, 0); //Airlab 111
  erode(temp1,contoursFrame, Mat());
}
```

Nella Figura 4.16 vengono riportati i risultati ottenuti mediante l'applicazione dell'algoritmo appena descritto.

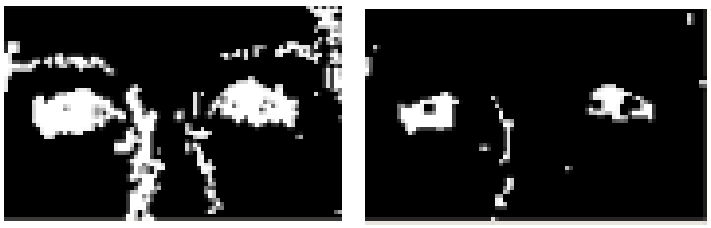

Figura 4.16: Confronto tra l'immagine binaria dell'area degli occhi (sx) e l'immagine erosa (dx).

Una volta completate le trasformazioni precedenti viene fatto nuovamente uso dell'algoritmo di analisi e riconoscimento dei contorni già introdotto durante la trattazione dei precedenti algoritmi di head e face detection.

In questo caso la selezione dei contorni relativi agli occhi è decisamente più complessa rispetto alla precedente. Nonostante le tecniche utilizzate, atte a rimuovere la componente principale di rumore dall'immagine, all'interno della regione d'interesse sono state trovate numerose aree che, in prima analisi, possono essere considerate come possibili occhi.

Per risolvere tale problematica, abbiamo introdotto un sistema di ricerca che permetta di eseguire un filtraggio tra tutti i possibili occhi, sfruttando alcune informazioni quali dimensioni, rapporto tra altezza e larghezza e posizionamento all'interno dell'immagine. I risultati ottenuti dall'algoritmo sono mostrati all'interno della Figura 4.17.

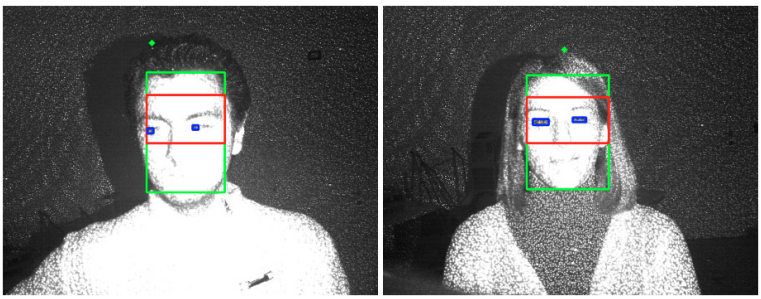

Figura 4.17: Risultati dell'algoritmo di eyes detection.

I risultati ottenuti mostrano come l'algoritmo permetta di estrarre la posizione e le dimensioni degli occhi indipendentemente dai tratti somatici e dalla posizione assunta dall'interlocutore. La presenza, inoltre, di lenti a contatto o occhiali da vista non compromette il risultato dell'analisi, come si può osservare dalla Figura 4.18.

Anche le variazioni delle condizioni di luce che si possono osservare durante l'arco della giornata, non influenzano in modo sostanziale i risultati prodotti dell'algoritmo il quale, come i precedenti, offre un rapido metodo di configurazione mediante un opportuno file di configurazione che ne permette l'ottimizzazione a determinate condizioni di luce.

L'algoritmo di riconoscimento dei movimenti

L'algoritmo di riconoscimento dei movimenti si discosta completamente dagli algoritmi di face detection ed eye detection precedentemente presentati; esso, infatti, si basa sull'implementazione della tecnica nota nel campo della

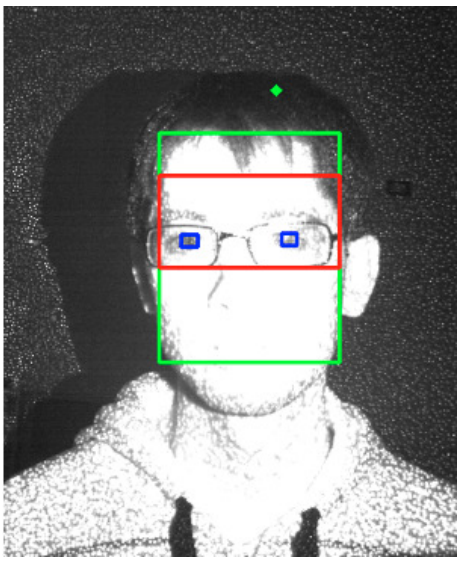

Figura 4.18: Risultati dell'algoritmo di eyes detection con un interlocutore con occhiali da vista.

visione artificiale con il nome di *Optical Flow*.

L'Optical FLow permette di individuare il campo delle velocità associato ai cambiamenti rilevati analizzando due immagini successive e tale risultato è dato proprio dal movimento relativo tra gli oggetti e la videocamera [21] (Figura 4.19).

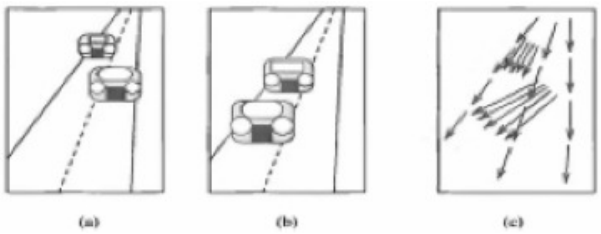

Figura 4.19: Algoritmo Optical Flow Sparso, (a) frame-1 (b) frame-2 (c) Optical Flow

L'obiettivo che ci si pone, dunque, consiste nel definire un valore che rappresenti la distanza percorsa da un pixel tra il frame precedente e quello corrente. Benché banale dal punto di vista teorico, l'implementazione di un'analisi di questo tipo (nota in letteratura come *Optical Flow Denso*) risulta computazionalmente molto complessa e, di conseguenza, si è optato per l'utilizzo della tecnica dell'*Optical Flow Sparso*.

Gli algoritmi che fanno parte di questa categoria si fondano sulla capacità di specificare, per ogni frame, particolari punti d'interesse, permettendo così di ottenere un tracking più agevole, robusto e preciso.

L'algoritmo di Optical Flow da noi elaborato, il cui schema è riportato nella Figura 4.20, ricorre all'utilizzo di due particolari algoritmi implementati all'interno delle librerie OpenCV, ossia:

- l'algoritmo *Shi-Tomasi corner detection*, per l'identificazione dei punti di interesse [31];
- l'algoritmo *Lucas-Kanade Optical Flow*, per effettuare il tracking dei punti di interesse individuati.

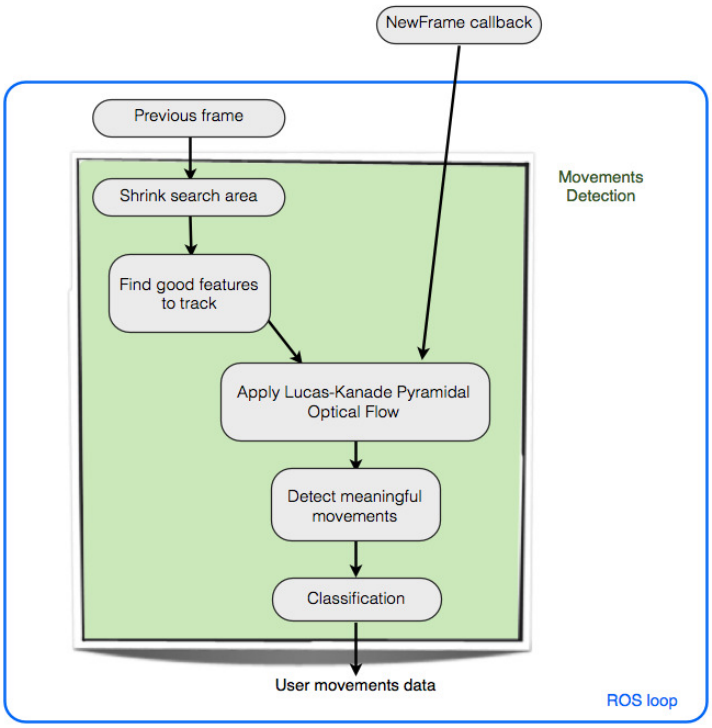

Figura 4.20: Architettura generale dell'algoritmo di riconoscimento dei movimenti

Come specificato in precedenza, gli algoritmi di Optical Flow si basano sull'analisi di sequenze di frame; operando in condizioni di real-time, quindi, è stato necessario prevedere un meccanismo di salvataggio del frame precedente e ritardare l'avvio dell'analisi dei movimenti al secondo frame ricevuto. Tale ritardo nell'avvio delle analisi non ha inficiato in alcun modo la qualità

delle stesse, bensì ha reso più semplice la sincronizzazione dei vari algoritmi utilizzati nel modulo di visione e più immediata la lettura dei risultati ottenuti.

Per ottimizzare i risultati generati dall'algoritmo di Optical Flow, inoltre, si è optato per applicare quest'ultimo unicamente sull'area degli occhi individuata dall'algoritmo di head detection, trattato in precedenza. La presenza di elementi facilmente distinguibili, quali occhi e sopracciglia, rende tale area la principale generatrice di informazioni utili per l'identificazione dei movimenti effettuati dall'utente. Tali elementi sono infatti sfruttati dall'algoritmo di Shi-Tomasi come punti di interesse; così facendo, inoltre, è stato possibile ridurre sia la quantità di dati da elaborare, rendendo l'algoritmo più agile, sia i possibili rumori generati da eventuali movimenti sullo sfondo.

Per l'individuazione dei punti di interesse, sul frame in memoria è stato applicato l'algoritmo di Shi-Tomasi, implementato nella libreria OpenCV mediante la funzione *goodFeaturesToTrack()*. L'algoritmo in questione non parte dalla definizione a priori di una buona zona da tracciare, ma si basa esclusivamente sull'algoritmo di tracking che si andrà ad utilizzare, in questo caso l'algoritmo di Lukas-Kanade. Applicato in tal contesto, l'algoritmo permette di estrarre le feature mediante i seguenti passi:

- calcolo del gradiente orizzontale e verticale dell'immagine o, come nel caso in analisi, dell'area selezionata mediante l'applicazione di una maschera;
- per ogni pixel, nell'intorno grande quanto la finestra d'integrazione della feature, determinazione degli autovalori minimi;
- ordinamento della lista di punti per valori decrescenti degli autovalori minimi;
- eliminazione dei punti troppo vicini (definizione di un parametro di vicinanza);
- selezione dei primi N punti utili.

dove il parametro di vicinanza ed il valore N sono specifiche di progetto.

Un esempio dei risultati ottenuti dall'applicazione di tale algoritmo è riportato in Figura 4.21

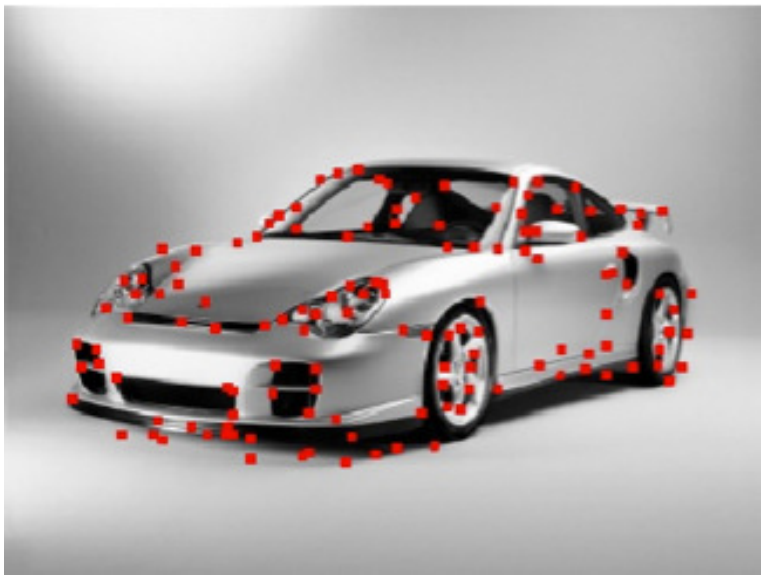

Figura 4.21: Esempio di estrazione dei punti d'interesse con l'algoritmo di Shi-Tomasi.

Andando ad eseguire alcune prove empiriche in relazione al caso di studi da noi analizzato, i parametri di progetto individuati per ottenere un algoritmo più performante sono:

- parametro di vicinanza: 0.1;
- numero massimo di punti d'interesse da evidenziare: 20.

mentre i restanti parametri richiesti dalla funzione hanno mantenuto il loro valore di default, consigliato all'interno delle librerie OpenCV.

```
//----- SHI & TOMASI ALGORITHM  -----
  //Saving of prevFrameIR_1C features
  vector<Point2f> prevFrameIRFeatures;
  //Mask setting
  Mat mask;
  CvPoint pt1Mask, pt2Mask;

  CvSize dim = cv::Size(frameIR.cols, frameIR.rows);
  mask = cv::Mat::zeros(dim, CV_8U);
  pt1Mask.x = eyesAreaBlob.getPt1().x + 10;
  pt1Mask.y = eyesAreaBlob.getPt1().y;
  pt2Mask.x = eyesAreaBlob.getPt2().x - 10;
  pt2Mask.y = eyesAreaBlob.getPt2().y;
```

```
//Computation of prevFrameIR_1C features
goodFeaturesToTrack(prevFrameIR_1C, prevFrameIRFeatures
    , FEATURES_NUMBER, .2, .1, mask, 3, false, 0.04);
```

Nella Figura 4.22 è possibile osservare i risultati ottenuti dall'applicazione di tale algoritmo.

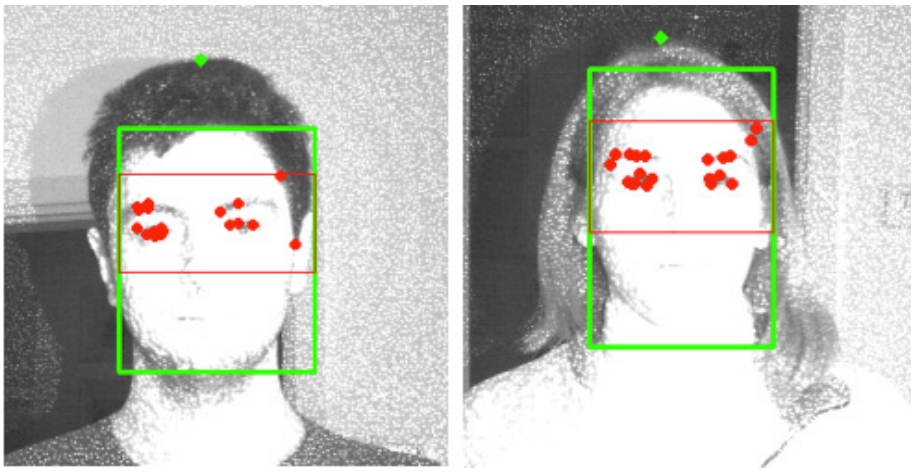

Figura 4.22: Risultati dell'algoritmo Shi-Tomasi.

Determinati i punti di interesse, siamo andati ad applicare l'algoritmo di *Lucas Kanade (LK)*. Il metodo di tracking sviluppato da LK opera allineando una finestra di pixel (di dimensioni ridotte) di due frame consecutivi, tramite una traslazione data da un offset. Ciò presuppone che le due immagini, acquisite a distanza di tempo ridotta (1 frame), non cambino eccessivamente e che i pixel degli oggetti presenti sulla scena mantengano comunque una forte relazione tra le due immagini.

Lo svantaggio di utilizzare piccole finestre locali, infatti, è dato proprio dal fatto che movimenti eccessivi (siano essi della camera o dell'utente) potrebbero portare i punti di interesse al di fuori dei margini della finestra d'interazione, rendendo così impossibile l'individuazione del movimento.

Tale problematica, però, è stata risolta mediante lo sviluppo dell'algoritmo di LK nella sua forma *piramidale*, ossia effettuando il tracking partendo dal livello più alto (minor dettaglio) di una piramide di immagini, per proseguire poi verso i livelli inferiori (dettaglio maggiore). L'utilizzo di questa particolare forma dell'algoritmo LK, dunque, permette di evitare la perdita di punti dovuta a movimenti ampi ed improvvisi.

Poiché l'elaborato in analisi si pone l'obiettivo di gestire l'interazione uomo-robot senza imporre particolari vincoli circa luminosità e struttura dell'ambiente circostante, per ottenere una maggiore robustezza del sistema di riconoscimento dei movimenti si è optato per l'utilizzo dell'algoritmo LK nella sua forma piramidale, ottenendo così i risultati mostrati nelle Figura 4.23.

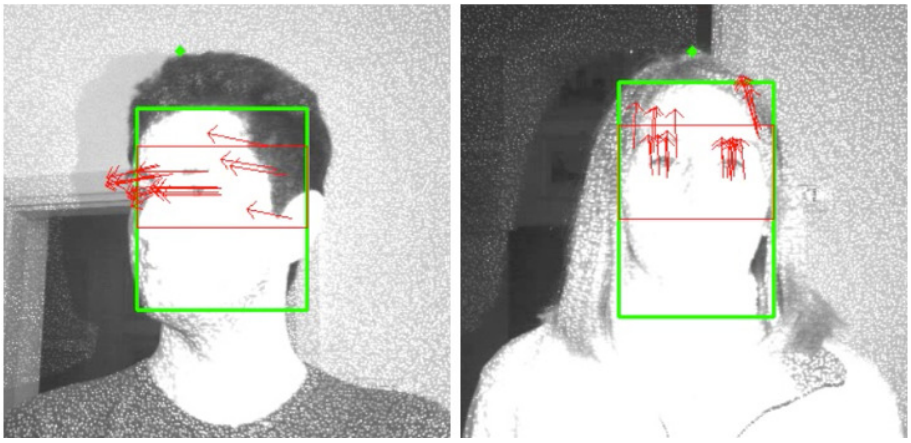

Figura 4.23: Risultati dell'algoritmo LK.

```
//----- LUCAS AND KANADE ALGORITHM  -----

  //Saving of frameIR_1C features
  vector<Point2f> frameIRFeatures;
  vector<uchar> foundFeatures;
  vector<float> featuresError;

  TermCriteria terminationCriteria = TermCriteria(
     CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, 20, .03);

  calcOpticalFlowPyrLK(prevFrameIR_1C, frameIR_1C,
              prevFrameIRFeatures, frameIRFeatures,
              foundFeatures, featuresError,
              cv::Size(15,15), 3, terminationCriteria,
                 0.5, 0);
```

Al fine di ottenere risultati più significativi, i dati generati dall'algoritmo LK vengono elaborati, permettendo così di ottenere un unico vettore identificativo del movimento effettuato dall'utente, come mostrato nella Figura 4.24.

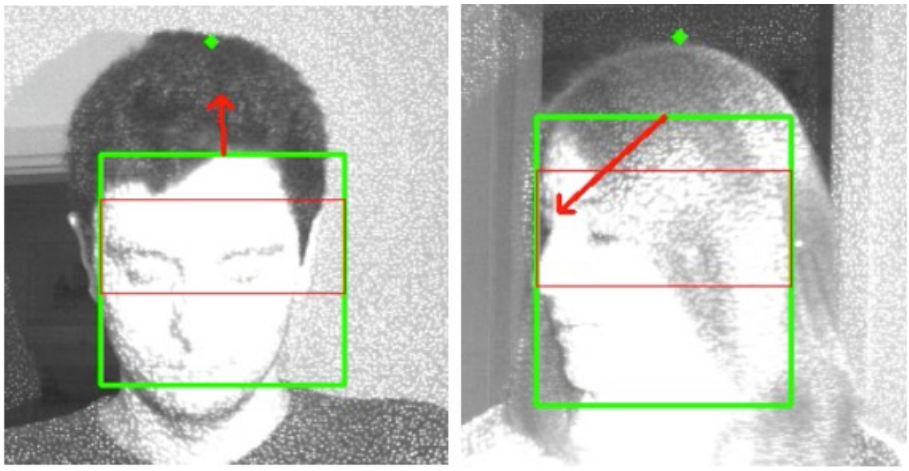

Figura 4.24: Risultati finali dell'algoritmo di Optical Flow.

A tal scopo, si è ricorsi al calcolo della mediana tra i punti d'interesse individuati su ciascun frame dell'algoritmo Shi-Tomasi; questa scelta ha permesso che alcuni punti d'interesse collocati ai bordi del volto (ad esempio in prossimità dei capelli) e classificabili come outlier, non inficiassero l'identificazione dei movimenti da parte dell'algoritmo.

Una volta determinato il punto mediano iniziale e finale del movimento, si è proceduto con il calcolo della sua direzione ed intensità mediante le seguenti relazioni matematiche (all'interno delle quali $pt1$ e $pt2$ rappresentano rispettivamente il punto iniziale e finale del vettore):

$$intensity = \sqrt{(pt1.y - pt2.y)^2 + (pt1.x - pt2.x)^2} \tag{4.4}$$

$$direction = \arctan \frac{(pt1.y - pt2.y)}{(pt1.x - pt2.x)} [rad] \tag{4.5}$$

La posizione finale del vettore, invece, è frutto di una semplice traslazione atta ad una migliore visualizzazione.

Il vettore così ottenuto rappresenta il punto di partenza per la definizione dei movimenti compiuti dall'utente; uno degli obiettivi che il progetto in analisi si pone, infatti, è quello di identificare lo stato d'interesse o disinteresse dell'interlocutore e per fare ciò il sistema non necessita della sola indicazione di movimento, ma deve saper riconoscere, data una sequenza di movimenti, quale sia l'azione compiuta dall'interlocutore.

In particolare, una serie di prove empiriche ci ha permesso di osservare come una determinata azione sia caratterizzata non tanto dall'intensità con cui vengono compiuti i movimenti che la compongono (differente da utente ad utente), quando dalle loro direzioni. Per tale ragione, abbiamo deciso di suddividere le direzioni dei movimenti calcolate tramite la relazione 4.5 in otto settori, in base alla rosa dei venti riportata in Figura 4.25 e di memorizzare, per ogni movimento, il settore di appartenenza della direzione calcolata.

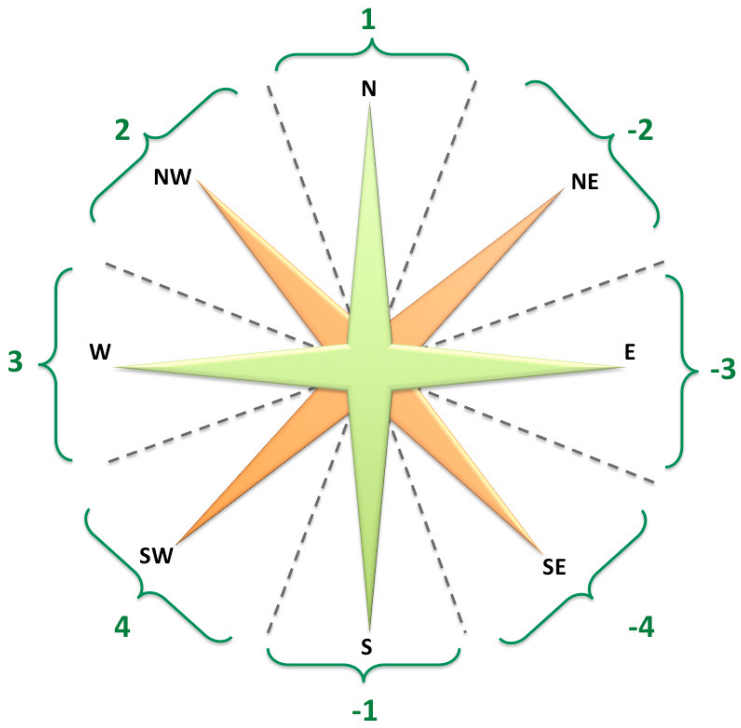

Figura 4.25: Rosa dei venti utilizzata per identificare il settore di appartenenza della direzione del movimento.

Tali valori vengono salvati all'interno di un opportuno vettore che rappresenta il punto di partenza per l'identificazione dell'azione compiuta dall'utente; a tale scopo, abbiamo deciso di ricorrere ad un algoritmo di classificazione che fosse in grado di determinare quale azione l'utente avesse compiuto, ricevendo in ingresso il vettore contenente l'identificativo del settore di appartenenza dei movimenti registrati a seguito di una determinata azione.

```
//Pruning
```

```
if ((move.size() >= MIN_MOVES) && (move.size() <=
    MAX_MOVES))
{
  moveIntensity = moveIntensity / move.size();

  //Interpolation
  interpolatedMoves = move.interpolate(
      INTERPOLATION_FACTOR, HERMITE_INTERPOLATION);

  //Saving interpolated vector
  for(int j = 0; j < INTERPOLATION_FACTOR; j++)
  {
    Point2DSample sample = Point2DSample(
        interpolatedMoves[j].getX(), j, 1.0, 'U');
    movement.insert(sample);
  }

  vector<Point2DSample> temp = movement.getSamples();
  userMoveClass = knn->getClassification(temp, knnScores,
      knnClass, KNN_WITH_RADIUS, 2.0, WITHOUT_LOG);
```

Prendendo come riferimento l'interazione uomo-uomo, si è scelto di focalizzare l'attenzione sulle seguenti quattro azioni:

- nod (annuire)
- deny (negare)
- right (voltarsi a destra)
- left (voltarsi a sinistra)

Interpolando i valori dei settori identificati nel corso di un passo interattivo e tracciando il relativo grafo (Figura 4.26), abbiamo potuto osservare come, in generale, i dati di ogni azione differiscano sufficientemente, rendendo quindi plausibile l'applicazione di un algoritmo di classificazione, quale l'algoritmo *K-nearest neighbor (KNN)*. La scelta è ricaduta su tale algoritmo in quanto esso mette a disposizione un sistema robusto e computazionalmente molto efficiente per classificare, in breve tempo, grandi quantità di dati.

Per l'implementazione dell'algoritmo KNN sono state utilizzate le librerie *Approximate Nearest Neighbor Searching (ANN)*, le quali mettono a disposizione numerose varianti dell'algoritmo in questione. Nel caso in analisi, si è scelto di utilizzare l'algoritmo *KNN with radius* il quale permette di limitare

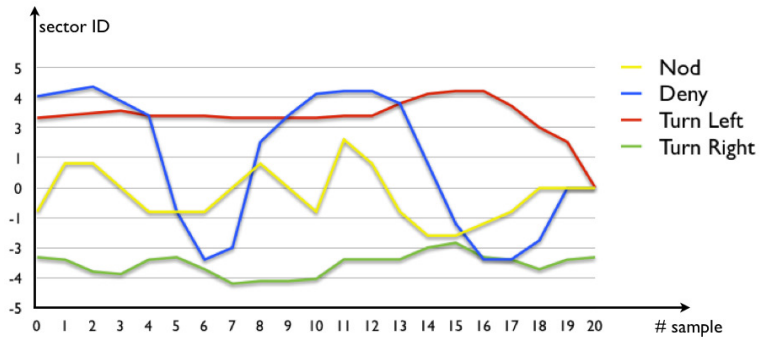

Figura 4.26: Esempio degli andamenti dei modelli di movimento appresi. In particolare, vengono qui riportati i risultati ottenuti dal processo di interpolazione, andando a specificare sull'asse delle ascisse il numero di campioni presi in considerazione, mentre sull'asse delle ordinate l'identificativo del settore di appartenenza del campione in esame.

facilmente l'intorno del punto all'interno del quale ricercare i K punti più vicini.

Una volta completate tutte le precedenti analisi, i risultati ottenuti vengono resi accessibili da tutti gli altri moduli del sistema tramite pubblicazione su tre differenti topic. A tale scopo si è resa necessaria la dichiarazione di tre differenti *publisher*, ognuno avente il compito di pubblicare le informazioni disponibili relative a volto, occhi e movimenti dell'interlocutore; tali informazioni verranno sfruttate all'interno del processo d'interazione come mostrato nel Capitolo 5.

```
pubHeadData = nh.advertise<head_analyzer::HeadDataMSG>("
    headData", 100);
pubEyesData = nh.advertise<head_analyzer::EyesDataMSG>("
    eyesData", 100);
pubMoveData = nh.advertise<head_analyzer::MoveDataMSG>("
    moveData", 100);
```

Per quanto riguarda i dati estratti dal volto dell'interlocutore, si è focalizzata l'attenzione su *pitch*, *roll* e *ratio* (Figura 4.27), tre parametri che risultano utili per l'identificazione del livello d'interesse manifestato dall'interlocutore durante l'interazione. In particolare, tali parametri sono determinati come segue:

- **Pitch**: tale parametro rappresenta l'inclinazione della testa rispetto al suo asse orizzontale, per tale ragione si è ritenuto opportuno, come mostrato in Figura 4.28, determinare il suo valore ricorrendo alla di-

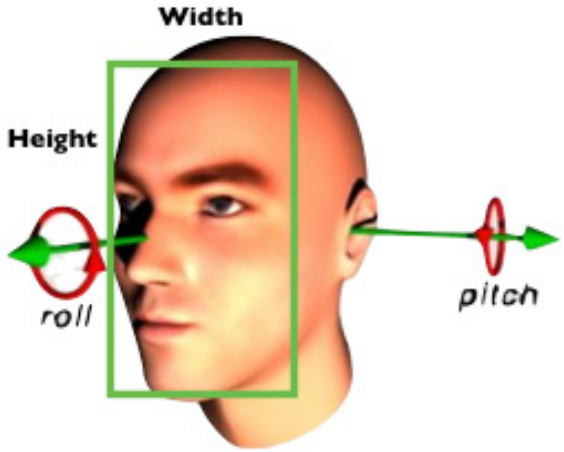

Figura 4.27: Indicazione dei parametri estratti dall'analisi del volto dell'utente, dove il ratio è espresso come: $ratio = height/width$

stanza esistente dal punto medio tra gli occhi, alla sommità del capo dell'utente.

- **Roll**: tale parametro rappresenta l'inclinazione della testa rispetto all'asse passante per il centro degli occhi. Come mostrato in Figura 4.29, esso è determinato come la differenza tra le coordinate verticali (y) del centro dei blob degli occhi.

- **Ratio:** tale parametro rappresenta il rapporto tra altezza e larghezza del ROI del volto dell'utente. Esso viene utilizzato come supporto o fonte alternativa d'informazione per il calcolo del roll, ad esempio nei casi in cui non vengano identificati gli occhi dell'utente.

Per quanto riguarda i dati estratti dall'analisi degli occhi, ci siamo concentrati sul rapporto (*ratio*) tra altezza e larghezza dei blob che li identificano. Tale informazione, infatti, risulta utile per determinare lo stato emotivo dell'utente durante l'interazione e, in particolar modo, il suo interesse nel proseguire o meno l'interazione.

Infine, per quanto riguarda il movimento dell'utente, si è scelto di pubblicare la classe riconosciuta dal processo di classificazione descritto in precedenza.

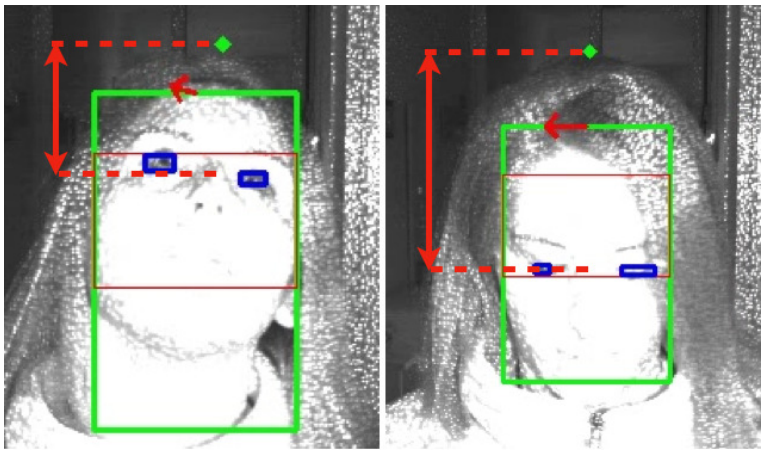

Figura 4.28: Calcolo dell'inclinazione del volto rispetto al suo asse orizzontale (pitch).

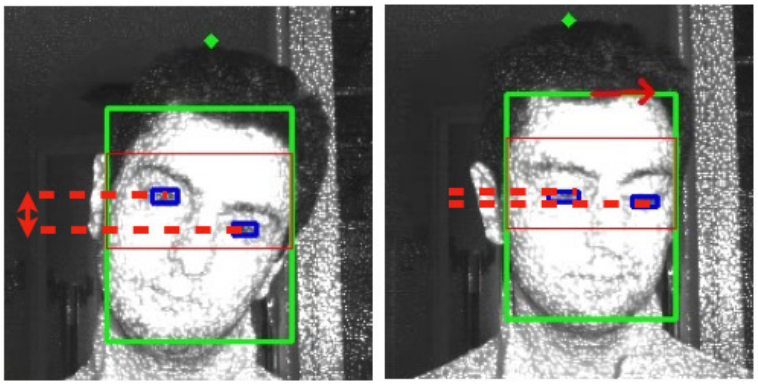

Figura 4.29: Calcolo dell'inclinazione del volto rispetto all'asse passante per il centro degli occhi (roll).

Capitolo 5

Il processo d'interazione

"Le emozioni non sembrano essere una simulazione molto utile per un robot... Non vorrei mai che il mio tostapane o l'aspirapolvere fossero così emotivi"

Agente Spooner (dal film - Io, Robot)

Numerose ricerche, come mostrato nel corso del Capitolo 2, hanno evidenziato l'importanza di progettare comportamenti quanto più possibili *human-like*, al fine di ottenere un'interazione efficace ed efficiente con un essere umano. Perché l'interlocutore umano sia effettivamente coinvolto nell'interazione, infatti, è indispensabile che il robot comunichi una propria intenzionalità, ossia riesca a far credere all'essere umano di aver opinioni, desideri ed intenzioni.

Lo sviluppo di un adeguato processo d'interazione, di conseguenza, ha rappresentato uno degli obiettivi principali del nostro elaborato ed ha richiesto un notevole sforzo per individuare il giusto bilanciamento tra la realizzazione di comportamenti sociali e l'implementazione di un sistema robusto ed efficiente.

In particolare, partendo dai dati forniti dai moduli *user-tracker* e *head-analyzer* e basandosi sugli studi d'interazione riportati in precedenza, abbiamo elaborato un sistema che permetta al robot di percepire il grado di interesse ed attenzione dell'interlocutore e di compiere, in base a questi, delle decisioni circa le modalità con cui mantenere o concludere l'interazione.

Il processo d'interazione è stato implementato e gestito interamente all'interno del modulo *e2-brain*, modulo ROS avente il ruolo di controllore

delle funzionalità interazionali e motorie del robot. L'architettura del modulo, riportata in Figura 5.1, evidenzia come il sistema sia concettualmente suddivisibile in due sezioni:

- **la gestione dei comportamenti:** sezione che si occupa della percezione dello stato d'interesse dell'utente, permettendo al robot di variare, di conseguenza, il suo comportamento;

- **la gestione dei movimenti:** sezione che si occupa degli spostamenti del robot nello spazio, controllando sia l'azionamento delle ruote, sia i dati forniti dalla cintura di sonar posta sulla base del robot stesso. Come verrà spiegato nel corso del capitolo, tali controlli hanno richiesto l'utilizzo di *MrBRIAN*, un software sviluppato dal Politecnico di Milano, atto all'implementazione ed alla gestione dei comportamenti di sistemi *behaviour based* mediante l'utilizzo della logica fuzzy [10] [9].

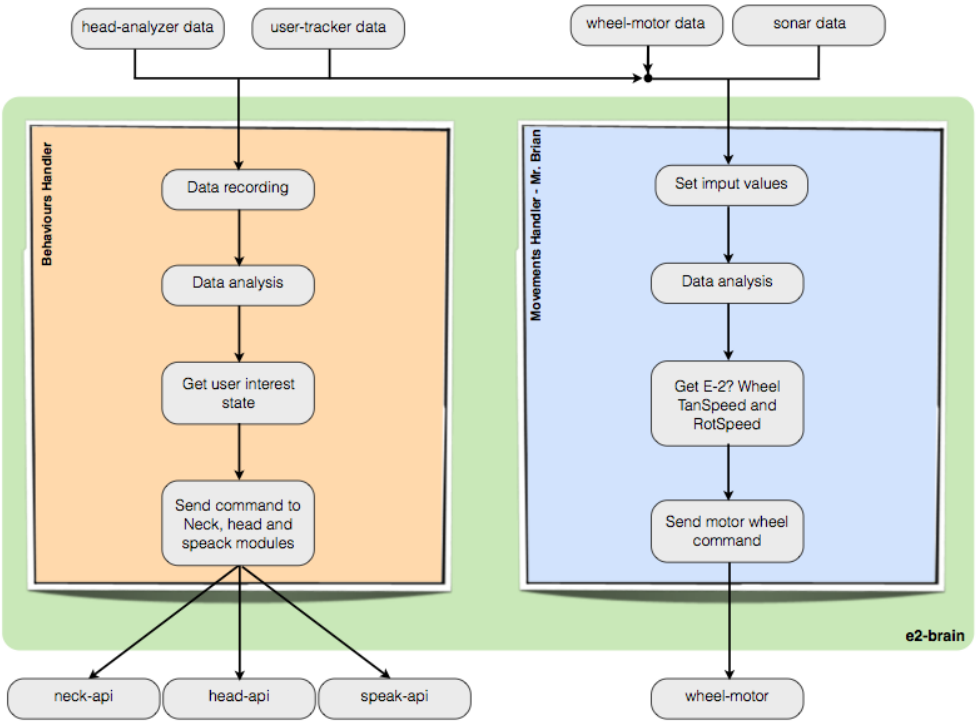

Figura 5.1: Architettura generale del modulo e2-brain.

5.1 Il modulo e2-brain: la gestione dei comportamenti

Si ricorre qui al concetto di *comportamento* nella sua accezione di "reazione allo stato d'animo percepito dall'utente"; come sottolineato in precedenza, infatti, i dati estratti dal sistema di visione hanno l'obiettivo di far comprendere al robot il grado d'interesse e di attenzione del suo interlocutore, permettendogli così di prendere delle decisioni circa la modalità con cui proseguire l'interazione.

La Figura 5.1 evidenzia come tale processo abbia inizio proprio con la ricezione dei dati ottenuti dai moduli *user-tracker* ed *head-analyzer* (illustrati nel corso del Capitolo 4), a seguito della quale il sistema procede con l'analisi dei dati raccolti, al fine di stabilire lo stato di coinvolgimento dell'interlocutore, permettendo così al robot di decidere come proseguire l'interazione in atto. La decisione elaborata viene infine attuata tramite appositi comandi inviati ai servomotori di collo, volto ed al modulo di sintesi vocale.

Le prime due parti del paragrafo focalizzeranno l'attenzione sulle varie fasi che compongono il processo d'interazione, mentre l'ultima parte andrà a definire nel dettaglio i moduli di servizio implementati per la realizzazione e la gestione dei comportamenti di E-2?.

5.1.1 Lettura ed analisi dei dati

Come osservato nel corso del Capitolo 4, il sistema di visione da noi implementato permette di ottenere informazioni circa i valori di *pitch*, *ratio* e *roll* della testa dell'utente, il *ratio* degli occhi, la classificazione del movimento compiuto ed, infine, la distanza uomo-robot.

Focalizzando per un istante l'attenzione sui valori di *pitch*, *ratio* e *roll* di testa ed occhi e sul valore della distanza uomo-robot, è importante sottolineare come, all'interno del processo d'interazione, essi non rivestano particolare rilevanza nel loro valore istantaneo, bensì risulta importante la variazione che essi subiscono. L'atto di inclinare lateralmente la testa, ad esempio, comunica un atteggiamento amichevole e ricettivo, ma per come è stato da noi determinato il valore di *roll*, non sarà il valore assunto da questo all'interno di un particolare frame ad identificare l'atto in questione, bensì il suo incremento e decremento in riferimento ad un periodo di tempo finito. Na-

sce quindi la necessità di memorizzare i dati per un determinato intervallo temporale, per poterne determinare poi, in fase di analisi, il corrispondente andamento.

A tal proposito, dagli studi di Kaliouby e Robinson [29] è emerso che il tempo necessario ad un essere umano per comprendere lo stato emotivo del proprio interlocutore sia pari a circa 2 secondi; si è dunque scelto di riportare tale tempistica del comportamento umano anche all'interno del processo cognitivo del robot, imponendo al sistema di procedere con la fase di analisi solo dopo aver collezionato dati per un minimo di 2 secondi (identificati, nel contesto in analisi, da un minimo di 60 campioni).

In alcuni casi, tuttavia, la durata del processo di salvataggio dei dati può essere estesa, al fine di permettere al robot di completare le attività definite dal comportamento attivato in precedenza; è importante sottolineare, infatti, che la fase di memorizzazione ed analisi dei dati corrisponde, lato utente, all'esecuzione di uno step dell'interazione e la possibilità di estendere l'intervallo di tempo dedicato a tale fase permette, ad esempio, la conclusione dell'esposizione, da parte del robot, di una determinata frase, prima che proceda con l'esecuzione del comportamento successivo.

Per quanto riguarda i dati relativi alla classificazione del movimento, invece, è stato riservato un trattamento leggermente differente: come per i dati precedenti, anche le classificazioni fornite dall'algoritmo di riconoscimento dei movimenti vengono qui memorizzate per il medesimo intervallo temporale riservato alla memorizzazione degli altri dati; in questo caso, però, la fase di analisi non sarà atta all'identificazione di una variazione dei valori registrati, bensì avrà il compito di identificare il movimento avente la maggiore frequenza all'interno dell'arco di tempo considerato.

Al fine di fornire una panoramica completa della fase di lettura e di analisi dei dati, vengono di seguito riportati i punti salienti relativi alla sua implementazione. La lettura dei dati computati dai moduli di visione, in accordo con il paradigma *publish and subscribe* utilizzato in ROS, ha richiesto l'iscrizione del modulo *e2-brain* ai *topic* d'interesse; tale operazione, come di consueto, ha richiesto l'utilizzo dei seguenti comandi:

```
//Messages subscribtions
ros::Subscriber subUserDistance = nh.subscribe("com", 10,
    getUserPositionData);
```

```
ros::Subscriber subUserHeadData = nh.subscribe("headData"
    , 10, getUserHeadData);
ros::Subscriber subUserEyesData = nh.subscribe("eyesData"
    , 10, getUserEyesData);
ros::Subscriber subUserMoveData = nh.subscribe("moveData"
    , 10, getUserMoveData);
ros::Subscriber subSonarData =  nh.subscribe("sonarData",
     10, getSonarData);
```

Una volta raggiunta la distanza d'interazione tra interlocutore e robot (che ricordiamo essere stata fissata, all'interno del modulo *head-analyzer*, ad un valore inferiore al metro), si avvia la fase di memorizzazione, durante la quale i dati vengono salvati all'interno di una particolare struttura, denominata *dataVectors*, contenente i vettori (uno per ciascun dato) entro i quali verranno salvati i 60 campioini raccolti. Il salvataggio dei dati all'interno dei corrispondenti vettori è reso possibile grazie all'utilizzo delle funzioni di callback, specificate all'interno dei subscriber.

```
//DataVectors
struct vectorStruct
{
  vector<int> userDistanceVect;
  vector<float> userHeadRatioVect;
  vector<int> userHeadPitchVect;
  vector<int> userHeadRollVect;
  vector<float> userEyeLRatioVect;
  vector<float> userEyeRRatioVect;
  vector<char> userMovesVect;
};

//Messages callbacks
void getUserPositionData(const user_tracker::Com com);
void getUserEyesData(const head_analyzer::EyesDataMSG
    eyes);
void getUserHeadData(const head_analyzer::HeadDataMSG
    head);
void getUserMoveData(const head_analyzer::MoveDataMSG
    move);
void getSonarData(const sonar::SonarData sonar);
```

Terminata la fase di memorizzazione dei dati, si procede con l'analisi di ogni singolo vettore contenuto all'interno della struttura *dataVectors*, con l'obiettivo di definire l'andamento predominante dei dati ivi contenuti. In particolare, per quanto riguarda i vettori contenenti i valori di *pitch*, *ratio* e *roll* della testa dell'utente, del *ratio* degli occhi ed i valori della distan-

za uomo-robot, è stata redatta la funzione *getVectorOutputValue()*, avente l'obiettivo di determinare l'andamento delle misure registrate (incremento, decremento, stazionarietà) rispetto all'iterazione precedente.

```
float getVectorOutputValue(vector<float>* vector)
{
      if (vector->size() > 0)
    temp = vector->at(0);
  else
    return result;

  for(int i = 1; i < vector->size(); i++)
    temp = temp + vector->at(i);

  average = temp /(vector->size());

  if(average > prevAverage) {result = 1.0; }
  else if (average < prevAverage) {result = -1.0; }
  else if (average == prevAverage) {result = 0.0; }

  return result;
}
```

Il vettore dei movimenti, invece, come detto poc'anzi, subisce un'analisi differente, atta a determinare il movimento percepito con maggiore frequenza all'interno della sessione temporale considerata; a tal proposito e stata implementa la funzione *detectMeaningfulMove()* sotto riportata.

```
char detectMeaningfulMove(vector<char>* vector)
{
  map<char, int> counter;
  map<char,int>::iterator it;

  pair<char,int> highest;
  highest.first = 'U';
  highest.second = 0;

  //put vector value into map
  for(int i = 0; i < vector->size(); i++)
  {
    if(counter.count(vector->at(i))>0)
    {
      it = counter.find(vector->at(i));
      (*it).second += 1;
    }
    else
```

```
    {
        counter.insert(pair<char,int>(vector->at(i),1));
    }
  }

  //go through map for finding max value
  for(it=counter.begin() ; it != counter.end(); it++)
  {
    //define max value
    if((*it).second > highest.second)
    {
      highest = make_pair((*it).first, (*it).second);
      cout<<"Highest: "<<highest.first<<" => "<<highest.
          second<<endl;
    }
    else if((*it).second == highest.second)
    {
      highest.first = 'U';
    }
  }

  return highest.first;
}
```

I risultati ottenuti dalle analisi sopra riportate rappresentano il punto di partenza per la gestione dell'interazione vera e propria, come verrà illustrato nel paragrafo seguente.

5.1.2 Gestione dell'interazione con E-2?

Diverse sono le tecniche illustrate in letteratura per la gestione dell'interazione uomo-robot: catene di Markov, reti dinamiche bayesiane, algoritmi probabilistici, etc; tuttavia, per gli obiettivi del presente elaborato, si è ritenuto opportuno ricorrere all'utilizzo di una struttura modellizzabile come un automa a stati finiti.

Ponendoci come obiettivo principale la realizzazione di un sistema che sia nel contempo robusto e caratterizzato da esigui tempi di elaborazione al fine di garantire un'interazione quanto più possibile *human-like*, infatti, abbiamo scelto di definire un insieme discreto di possibili stadi in cui l'interazione possa evolversi, così come si è optato per l'individuazione di due soli possibili stati d'animo dell'utente: *interessato* e *non-interessato*. Sarà proprio in base al livello di interesse percepito che il robot deciderà la mo-

dalità con cui evolvere l'interazione in corso.

L'automa a stati finiti, nel suo essere modello di un sistema dinamico, invariante e discreto, ben si presta alla realizzazione del progetto d'interazione da noi ideato: ricorrendo allo stato d'animo percepito come segnale d'ingresso e considerando le fasi dell'interazione come stati, infatti, è stato possibile ottenere un sistema che evolvesse nel tempo passando da uno stato all'altro dell'interazione in funzione del livello d'interesse manifestato dall'utente.

Consapevoli dei limiti di adattabilità dati da un sistema così concepito, abbiamo cercato di garantire una maggiore flessibilità andando a definire l'automa mediante un file di configurazione, permettendo così di applicare anche grandi variazioni al processo d'interazione senza che ciò richieda alcun intervento diretto sul codice e/o ricompilazione del modulo in esame.

Ogni stato d'animo del sistema, infine, avrà al suo interno le informazioni necessarie per l'attuazione della fase d'interazione che rappresenta; a tal proposito sono state realizzate una serie di primitive di alto livello che, attivate di volta in volta in base allo stato in cui si trova, permettono al robot di assumere un particolare comportamento, andando, ad esempio, a comandare i servomotori del volto, del collo o a specificare la frase che il robot E-2? dovrà riprodurre.

Andiamo ora a fornire una panoramica più esaustiva del processo d'interazione da noi ideato; come evidenziato poc'anzi, tale processo si basa sulla percezione, da parte del robot, di due particolari stati d'animo dell'utente: *interessato* o *non interessato*. L'individuazione di queste caratteristiche è ottenuta mediante un meccanismo di punteggi assegnati a ciascuno stato, sulla base dei risultati ottenuti dalla precedente fase di lettura ed analisi dei dati forniti dal sistema di visione.

Come evidente dalla Figura 5.2, infatti, ciascuna delle feature individuate dal sistema di visione è in grado di influire sull'indentificazione del grado di interesse manifestato dall'utente, andando ad assegnare, di conseguenza, un punteggio a ciascuno dei due stati d'animo presi in considerazione. Il peso di tali punteggi è il frutto della combinazione tra la ricerca effettuata nel campo del linguaggio del corpo ed una serie di prove empiriche da noi effettuate in laboratorio in corso d'opera.

Come evidenziato in precedenza, ciascuna fase dell'interazione è stata

VARIABLE	VALUE		STATUS MEANING	
			INTERESTED	*NOT INTERESTED*
userDistance: system analyses difference of distance between user and robot during 60 frame and decides if user is closer, farther or still	Farther	1	-1	1
	Steady	0	1	-1
	Closer	-1	2	-2
HeadPitch: related on distance between Y-coord average of eyes and *userHeadPosition*.	Increase (head-down)	1	-2	2
	Decrease (head-up)	-1	2	-2
	Steady	0	1	1
HeadRoll: related on difference between Y-coord of each eye-blob center.	Tilted	1	2	-2
	Centered	0	1	0
EyesRatio: defined the average of eye dimension ratio (literature), system can detect if user's eyes ratio is bigger or smaller than ratio into the previous frame	Bigger	1	1	-1
	Steady	0	0	0
	Smaller	-1	-1	1
Move Detected	Nod	N	3	-2
	Deny	D	-2	3
	Turn Left	L	-1	2
	Turn Right	R	-1	2

Figura 5.2: Tabella di punteggi per l'individuazione del grado di interesse dell'utente.

da noi ideata come lo stato di un automa a stati finiti; per fare ciò si è resa necessaria l'implementazione di due particolari librerie: *Node.h* e *StateMachine.h*. La libreria *Node.h* si occupa di specificare le caratteristiche proprie di un singolo stato, caratterizzato dai seguenti elementi:

- **ID:** identificativo univoco dello stato;
- **Name:** nome (univoco) dello stato, utile nel comprendere la fase d'interazione in atto;
- **Output:** variabile d'uscita dello stato. Tale parametro viene utilizzato per specificare le primitive di alto livello che devono essere attivate per attuare il comportamento desiderato dalla fase d'interazione raggiunta. L'output, in particolare, è composto da una stringa binaria di 10 bit; ogni bit è stato associato ad una particolare primitiva, che potrà essere così chiamata ponendo ad 1 il relativo bit;
- **Params:** alcune primitive di alto livello necessitano della specifica di vari parametri; la stringa *params* contiene, quindi, tutti i parametri necessari per la corretta attivazione delle primitive specificate nella stringa di output. La sintassi da noi utilizzata è la seguente:

$$par11, ..., par1N : par21, ...par2N : ... : parM1, ..., parMN$$

- **Next:** elenco dei noti direttamente raggiungibili dal nodo corrente.

Per quanto riguarda la libreria *StateMachines.h*, invece, essa permette la definizione dell'automa come vettore di stati opportunamente connessi, garantendone la corretta evoluzione a partire da uno stato iniziale ed un segnale di ingresso. È importante sottolineare come tale libreria non si occupi della definizione degli stati veri e propri ideati per l'interazione, ma solo della loro gestione; come evidenziato in precedenza, infatti, la definizione degli stati avviene all'interno di un apposito file di configurazione, collocato all'interno della cartella *config* del modulo *e2-brain*, al fine di offrire una maggiore flessibilità del sistema.

Riportiamo di seguito un esempio del processo d'interazione da noi ideato per effettuare i test del sistema complessivo: si consideri di voler attuare l'interazione modellizzata dell'automa riportato in Figura 5.3; si osservi come, all'interno di ogni nodo, siano indicate le primitive da attivare (*speak, neck e head*) e, per ciascuna di esse, il parametro indicativo della relativa

frase da pronunciare, espressione da manifestare e movimento del collo da attuare.

Tale mappa d'interazione è ottenuta mediante la definizione del file di configurazione riportato di seguito, all'interno del quale su può osservare come le primitive selezionate corrispondano rispettivamente ai bit occupanti la posizione 1°, 3° e 5° della stringa di output e come ciascuna di queste richieda la specifica di uno o più parametri necessari per la sua attuazione.

```
# The first part cointains all state machine node,
   defined as
# ID; NODE_NAME; OUTPUT; PARAMS; (first node = initial
   node)
0;  START;            0000000000;     :;
1;  APPROACH;         1010100000;     0:1,10:5:;
2;  START_INT_1;      1010100000;     1:0:0:;
3;  START_INT_2;      1010100000;     1:1,-10:5:;
4;  EXPLAIN_1;        1010100000;     3:0:2:;
5;  EXPLAIN_2;        1010100000;     4:0:5:;
6;  EXPLAIN_3;        1010100000;     5:1,-20:2:;
7;  EXPLAIN_4;        1010100000;     4:0:0:;
8;  EXPLAIN_5;        1010100000;     6:1,-10:2:;
9;  INVITE_1;         1010100000;     6:3,4:1:;
10; REGARDS_1;        1010100000;     2:1,20:3:;
11; REGARDS_2;        1010100000;     8:1,-15:0:;
12; REGARDS_3;        1010100000;     7:0:5:;
---
# The second part contains all transaction between nodes
   defined as
# START_NODE_NAME;ENDING_NODE_NAME;(int)TRANSACTION_VALUE
     enum transaction { INTERESTED, NOT_INTERESTED };
START;                APPROACH;               0;
START;                APPROACH;               1;
APPROACH;             START_INT_2;            0;
APPROACH;             START_INT_1;            1;
START_INT_2;          EXPLAIN_1;              0;
START_INT_2;          EXPLAIN_4;              1;
START_INT_1;          EXPLAIN_4;              0;
START_INT_1;          REGARDS_1;              1;
EXPLAIN_1;            EXPLAIN_2;              0;
EXPLAIN_1;            EXPLAIN_4;              1;
EXPLAIN_4;            EXPLAIN_3;              0;
EXPLAIN_4;            REGARDS_1;              1;
EXPLAIN_2;            EXPLAIN_3;              0;
EXPLAIN_2;            EXPLAIN_5;              1;
```

```
EXPLAIN_3;              INVITE_1;               0;
EXPLAIN_3;              INVITE_1;               1;
EXPLAIN_5;              REGARDS_3;              0;
EXPLAIN_5;              REGARDS_2;              1;
INVITE_1;               REGARDS_3;              0;
INVITE_1;               REGARDS_2;              1;
---
```

All'avvio, il modulo *e2-brain* andrà quindi a caricare tale file di configurazione

```
//Load state machines from configuration file
StateMachine *sm = new StateMachine();
string stateMachineConfigFile = ros::package::getPath("
    e2_brain")+"/config/state_machine_config/
    state_machine.txt";
err = sm->loadFromFile(stateMachineConfigFile);
sm->printMap();
```

e ad ogni ciclo dell'algoritmo, dopo aver elaborato il grado di interesse dell'utente, si occuperà di gestire l'evoluzione dell'automa mediante l'invocazione del metodo *run()*, avente, come parametro d'ingresso, lo stato d'interesse percepito e, come parametro d'uscita, il nuovo stato raggiunto all'interno dell'automa specificato.

```
node = sm->run(t);
if(node != NULL)
{
  //Print node info
  node->displayNode();

  //Check state machines output and send goals to server
  primitiveMSGParser(node->getOutput(), node->getParams()
      );
}
```

Il metodo *primitiveMSGParser()* si occuperà, infine, di decodificare le stringhe *output* e *params* proprie del nuovo nodo raggiunto dall'automa e richiamare le corrispondenti primitive di alto livello per l'attuazione del comportamento selezionato.

5.1.3 Attuazione dei comportamenti di E-2?

Come annunciato in precedenza, l'attuazione dei comportamenti di E-2? avviene tramite una chiamata ad opportune primitive di alto livello, ciascuna

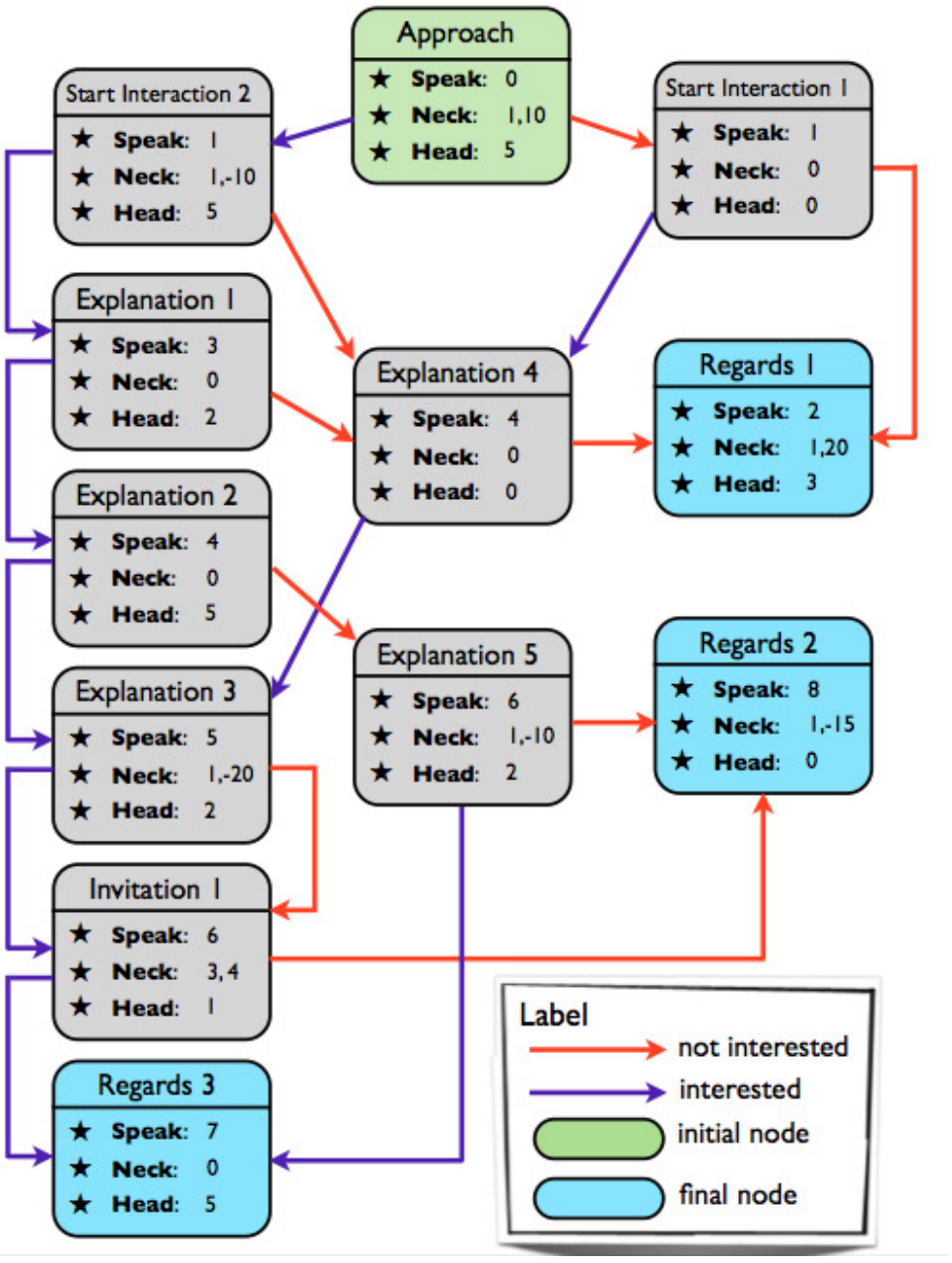

Figura 5.3: Esempio di automa a stati elaborato per il processo d'interazione di E-2?.

delle quali si occuperà della realizzazione del proprio task (espressione del volto, movimento del collo, gestione del software di sintesi vocale, etc.).

Tali primitive sono state implementate ricorrendo ad opportuni moduli di servizio, ognuno dei quali è stato realizzato come singolo modulo ROS posto all'interno dello stack *interaction-api*. Ogni modulo è stato concepito sfruttando le direttive delle librerie *actionLib* (introdotte nel corso del Capitolo 3), le quali hanno permesso l'utilizzo del paradigma client-server per una corretta gestione delle richieste multiple inviate dal modulo *e2-brain*.

In particolare, per la realizzazione degli obiettivi preposti dal presente elaborato, abbiamo proceduto all'implementazione dei seguenti tre moduli di servizio:

- **head-api:** modulo che si occupa della messa in funzione dei servomotori presenti all'interno della testa di E-2?, al fine di dare al robot un'espressione coerente con la fase d'interazione in atto;
- **neck-api:** modulo che si occupa di mettere in servizio i servomotori presenti sulla struttura meccanica adibita a collo di E-2?, al fine di fare assumere il robot diverse pose nel corso dell'interazione;
- **speak-api:** modulo che si occupa della gestione del software di sintesi vocale utilizzato. Nel contesto in analisi, in particolare, si è scelto di ricorrere al software open source *eSpeak*, già utilizzato nelle versioni precendenti di E-2?.

L'architettura generale del sistema di attuazione dei comportamenti è riportata in Figura 5.4 e la sua inizializzazione avviene direttamente all'interno del modulo *e2-brain*, il quale si occuperà di inizializzare i processi client e connettere questi ultimi con i corrispondenti moduli server in esecuzione.

```
//Action clients definition
typedef actionlib::SimpleActionClient<speak_api::
   SpeakAction> SpeakClient;
typedef actionlib::SimpleActionClient<head_api::
   HeadAction> HeadClient;
typedef actionlib::SimpleActionClient<neck_api::
   NeckAction> NeckClient;

//Clients definition
SpeakClient speakClient("speak_api", true);
HeadClient headClient("head_api", true);
NeckClient neckClient("neck_api", true);
```

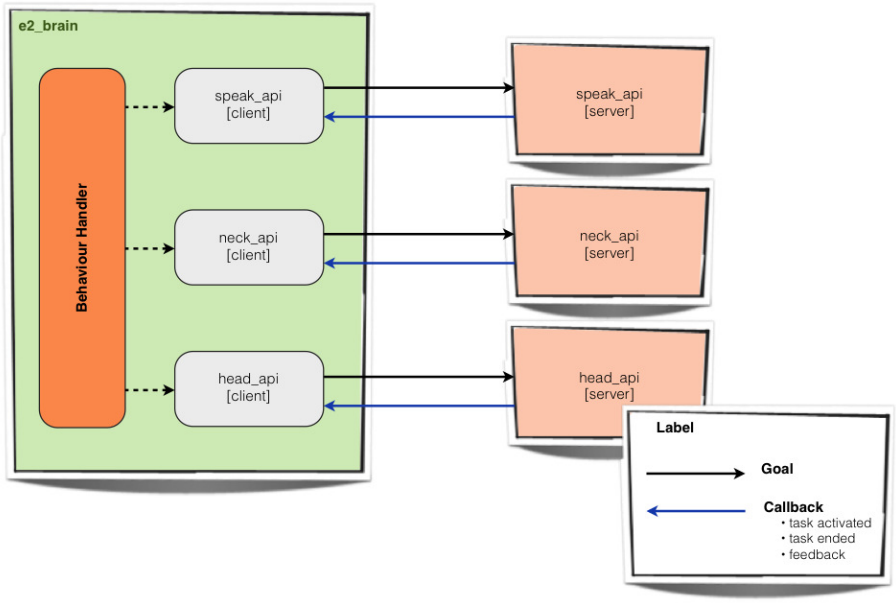

Figura 5.4: Architettura generale del sistema di attuazione dei comportamenti.

```
//Wait for servers
speakClient.waitForServer();
headClient.waitForServer();
neckClient.waitForServer();
```

La comunicazione tra i processi client e moduli server avviene tramite la definizione delle azioni messe a disposizione da ciascun server e tramite apposite funzioni di callback. Per quanto riguarda le azioni, in particolare, esse sono specificate all'interno di un opportuno file *NomeAzione.action*, sfruttando una semantica simile a quella vista per i messaggi ROS. All'interno del file l'azione viene definita mediante la specifica delle tre seguenti sezioni:

- **Goal:** in questa sezione sono definite tutte le variabili da utilizzare per specificare l'azione che dovrà essere compiuta dal server;

- **Result:** in questa sezione sono definite le variabili entro cui il server dovrà fornire gli output una volta terminata l'azione in oggetto. In relazione all'elaborato in analisi, in particolare, si è scelto di fornire in uscita un semplice codice di controllo per verificare a runtime il risultato dell'esecuzione del task;

- **Feedback:** in questa sezione sono definite le variabili i cui valori verranno periodicamente inviati al client, al fine di controllare lo stato di avanzamento del task in esecuzione.

Per quanto concerne le funzioni di callback, invece, la loro definizione avviene all'interno del modulo *e2-brain* ed hanno l'obiettivo di permettere ai moduli server di comunicare al sistema l'avvenuta attivazione dell'azione richiesta, i feedback desiderati ed il completamento dell'azione stessa.

```
//Server action done callbacks
void speakDoneCallback(const actionlib::
   SimpleClientGoalState& state,
                       const speak_api::
                          SpeakResultConstPtr& result);
void headDoneCallback(const actionlib::
   SimpleClientGoalState& state,
                       const head_api::HeadResultConstPtr
                          & result);
void neckDoneCallback(const actionlib::
   SimpleClientGoalState& state,
                       const neck_api::NeckResultConstPtr
                          & result);
void wheelDoneCallback(const actionlib::
   SimpleClientGoalState& state,
                       const wheel_motor::
                          WheelResultConstPtr& result);
void kinectDoneCallback(const actionlib::
   SimpleClientGoalState& state,
                       const kinect_motor::
                          KinectResultConstPtr& result);
//Server action activated callbacks
void speakActiveCallback();
void headActiveCallback();
void neckActiveCallback();
void wheelActiveCallback();
void kinectActiveCallback();
//Server action feedBack callback
void speakFeedbackCallback(const speak_api::
   SpeakFeedbackConstPtr& feed);
void headFeedbackCallback(const head_api::
   HeadFeedbackConstPtr& feed);
void neckFeedbackCallback(const neck_api::
   NeckFeedbackConstPtr& feed);
```

```
void wheelFeedbackCallback(const wheel_motor::
    WheelFeedbackConstPtr& feed);
void kinectFeedbackCallback(const kinect_motor::
    KinectFeedbackConstPtr& feed);
```

L'attivazione di una particolare azione è gestita direttamente all'interno del modulo *e2-brain*, il quale ha, tra gli altri, il compito di inviare il *goal* dell'azione in esame al particolare server di riferimento; nell'effettuare tale chiamata, il modulo *e2-brain* andrà a specificare anche le tre funzioni di callback da utilizzare.

```
//Speaking
if(activationAPI.apiCode[SPEAK] == 1)
{
  activationAPI.apiCode[SPEAK] = -1;
  speakHandlerFree = false;

  //Get phrase number
  stringstream ss(activationAPI.apiPar[SPEAK]);
  int temp;
  ss >> temp;

  //Define goal and send it to the server side
  speakGoal.speed = phrases[temp].speed;
  speakGoal.language = phrases[temp].language;
  speakGoal.phrase = phrases[temp].data;
  speakClient.sendGoal(speakGoal, &speakDoneCallback, &
      speakActiveCallback, &speakFeedbackCallback);
}

//Head expression
if(activationAPI.apiCode[HEAD] == 1)
{
  activationAPI.apiCode[HEAD] = -1;
  headHandlerFree = false;
  int temp;
  //Get head action code (first value of params)
  size_t found = 0;
  found = activationAPI.apiPar[HEAD].find_first_of(",");
  if(found > 0)
  {
    string activationCode = activationAPI.apiPar[HEAD].
        substr(0, found);
    stringstream ss(activationCode);
    ss >> temp;
    //Remove fiurst params to string
```

```
    activationAPI.apiPar[HEAD].erase(0, found + 1);

    //Set goal code
    headGoal.actionCode = temp;
  }
  else
  {
    stringstream ss(activationAPI.apiPar[HEAD]);
    ss >> temp;
    //Set goal code
    headGoal.actionCode = temp;
  }

  //Define goal and send it to the server side
  headGoal.params = activationAPI.apiPar[HEAD];
  headClient.sendGoal(headGoal, &headDoneCallback, &
     headActiveCallback, &headFeedbackCallback);
}

//Neck moves
if(activationAPI.apiCode[NECK] == 1)
{
  activationAPI.apiCode[NECK] = -1;

  neckHandlerFree = false;
  int temp;
  //Get head action code (first value of params)
  size_t found = 0;
  found = activationAPI.apiPar[NECK].find_first_of(",");
  if(found > 0)
  {
    string activationCode = activationAPI.apiPar[NECK].
       substr(0, found);
    stringstream ss(activationCode);
    ss >>temp;
    //Remove first params to string
    activationAPI.apiPar[NECK].erase(0, found + 1);

    //Set goal code
    neckGoal.actionCode = temp;
  }
  else
  {
    stringstream ss(activationAPI.apiPar[NECK]);
    ss >> temp;
```

```
    //Set goal code
    neckGoal.actionCode = temp;
  }

  //Define goal and send it to the server side
  neckGoal.params = activationAPI.apiPar[NECK];
  neckClient.sendGoal(neckGoal, &neckDoneCallback, &
      neckActiveCallback, &neckFeedbackCallback);

}
```

Si procede ora andando ad illustrare nel dettaglio come sono stati implementati i tre moduli di servizio da noi utilizzati; a tal proposito, è importante evidenziare che, al fine di non incrementare eccessivamente la complessità del sistema, si è preferito predisporre un'unica azione possibile per ciascun modulo server.

Il modulo head-api

Come evidenziato in precedenza, il modulo *head-api* si occupa della messa in funzione dei servomotori presenti all'interno della testa di E-2?, al fine di dare al robot un'espressione coerente con la fase d'interazione in atto. La definizione dell'azione di tale modulo è riportata all'interno del file *Head.action*, contenuto all'interno della cartella *action* del modulo in analisi.

```
#Goal definition
int32 actionCode
string params
---
#Results definition
int32 status
---
#Feedback definition
uint8 feedback
```

Come si può osservare dal dettaglio implementativo riportato sopra, il *goal* di tale azione è definito ricorrendo a due variabili:

- **actionCode:** tale variabile viene utilizzata dal modulo *e2-brain* per definire quale API di alto livello dovrà essere eseguita, ricorrendo ai dati in uscita dal processo di gestione dei comportamenti analizzato in precedenza;
- **params:** per far si che i servomotori del volto di E-2? raggiungano una determinata posizione, le primitive di alto livello attivate possono

richiedere alcuni parametri. Tali parametri possono essere comunicati al server mediante questa variabile stringa, seguendo la sintassi illustrata in precedenza per la variabile *params* definita all'interno della classe *Node.h*.

Le restanti due sezioni dell'azione implementata per il modulo in esame permettono al server di comunicare al processo client sia alcuni valori di controllo (*feedback*), sia l'avvenuto completamento dell'azione richiesta (*status*).

Per quanto riguarda le primitive di alto livello realizzate per il modulo in esame, infine, si è optato per far si che ogni primitiva corrispondesse ad una particolare espressione del robot e, in particolare, sono state definite le seguenti espressioni (Figura 5.5):

- **Standard (cod.0):** espressione neutra di E-2?, utilizzata anche come posizione di partenza per i servomotori del volto;
- **Invitation (cod.1):** espressione utilizzata nella fase di invito all'interlocutore a seguire il robot verso un particolare punto di interesse (nel contesto di una fiera, ad esempio, verso uno specifico stand);
- **Interested (cod.2):** espressione utilizzata quando il robot interagisce con un interlocutore particolarmente interessato;
- **Angry (cod.3):** espressione utilizzata quando il robot sceglie di terminare anticipatamente l'interazione, a seguito del disinteresse riscontrato nell'interlocutore;
- **Annoyed (cod.4):** espressione utilizzata quando il robot percepisce un calo d'interesse durante l'interazione;
- **Happy (cod.5):** espressione utilizzata quando il robot percepisce un elevato grado d'interesse ed attenzione del proprio interlocutore.

Il modulo neck-api

Il modulo *neck-api*, come annunciato in precedenza, si occupa di mettere in servizio i servomotori presenti sulla struttura meccanica adibita a collo di E-2?, al fine di far assumere al robot diverse pose nel corso dell'interazione. La definizione dell'azione di tale modulo è riportata all'interno del file *Neck.action*, contenuto nella cartella *action* del modulo in analisi.

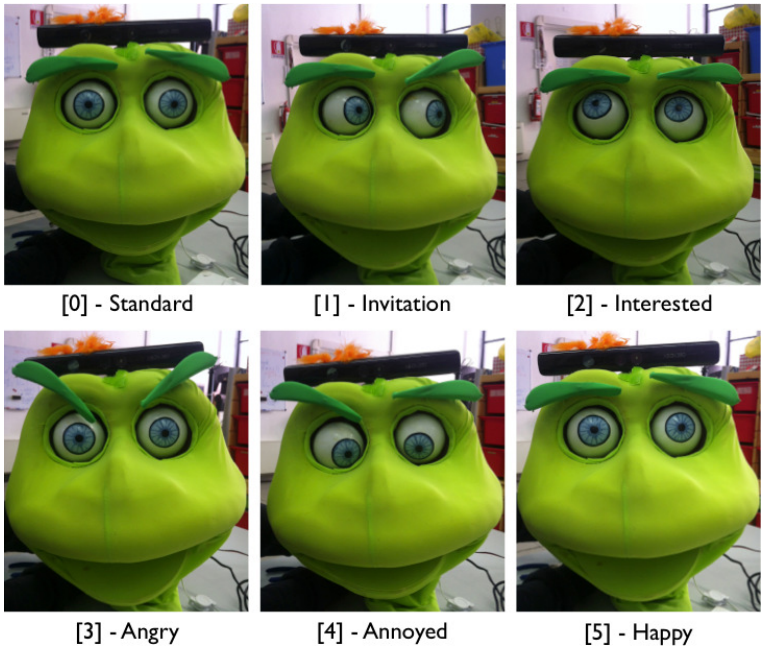

Figura 5.5: Espressioni implementate del volto del robot E-2? .

```
#Goal definition
int32 actionCode
string params
---
#Results definition
int32 status
---
#Feedback definition
uint8 feedback
```

Dal dettaglio implementativo sopra riportato, risulta evidente come tale modulo si basi sul medesimo meccanismo di funzionamento illustrato per il modulo *head-api*, tant'è che le due azioni definite per tali moduli corrispondono.

Ovviamente differenti, invece, sono le primitive di alto livello implementate che, per il modulo in analisi, hanno il compito di comunicare a ciascun servomotore installato la variazione che deve effettuare, rispetto alla posizione iniziale, per ottenere la posa definita. In particolare, per il collo di E-2? sono stati definiti i seguenti movimenti:

- **Reset (cod.0):** primitiva rappresentante la posizione di partenza dei servomotori del collo, coincidente con la posizione eretta dello stesso;

- **Tilt-left-or-right (cod.1):** primitiva che permette di inclinare il collo del robot rispetto al suo asse verticale, a destra o a sinistra in base al valore dei parametri in ingresso;

- **Tilt-ahead-or-back (cod.2):** primitiva che permette di inclinare il collo del robot rispetto al suo asse orizzontale, in avanti o indietro in base al valore dei parametri passati in ingresso;

- **Bow (cod.3):** primitiva che permette di simulare il movimento di un inchino, avente l'obiettivo di indicare all'utente la direzione verso cui il robot vuole accompagnarlo; le diverse tipologie di inchino possono essere selezionate sulla base dei parametri passati in ingresso alla primitiva.

Il modulo speak-api

Il modulo *speak-api*, come annunciato in precedenza, si occupa della gestione del software di sintesi vocale utilizzato. La definizione dell'azione di tale modulo è riportata all'interno del file *Speak.action*, contenuto nella cartella *action* del modulo in analisi.

```
#Goal definition
string phrase
string language
string speed
---
#Results definition
int32 status
---
#Feedback definition
uint8 feedback
```

Come si può osservare dal dettaglio implementativo sopra riportato, il *goal* di tale azione è definito ricorrendo a tre variabili:

- **phrase:** tale variabile viene utilizzata per indicare la frase che il sintetizzatore vocale deve riprodurre;

- **languade:** tale variabile viene utilizzata per specificare la lingua di riferimento per enunciare la frase indicata in precedenza;

- **speed:** tale variabile permette di specificare la velocità di lettura della frase, espressa in parole al minuto.

Le restanti due sezioni dell'azione sopra riportata, invece, coincidono esattamente con quanto illustrato in relazione ai due moduli precedenti.

Coerentemente con le scelte progettuali da noi effettuate in relazione all'implementazione dell'automa a stati finiti, per mantenere il più elevato possibile il livello di flessibilità dell'algoritmo, anche in questo caso si è scelto di non definire le possibili frasi da pronunciare all'interno del modulo in analisi, bensì di ricorrere ad un apposito file di configurazione, situato nella cartella *config* del modulo *e2-brain*, all'interno del quale ciascuna frase è specificata secondo la seguente sintassi:

$$speed : language :' phrase'$$

Ricollegandosi all'esempio d'interazione riportato nel paragrafo precedente, la Figura 5.6 riporta un ipotetico file di configurazione per il sistema di sintesi vocale.

```
160:it:'ciao, io sono i-tuu. Se ti và, ti racconto qualcosa di mé.'
150:it:'Sono stato progettato dal politècnico di Milano; i miei progettìsti mi hanno
    definito un robot da intrattenimento'
150:it:'Se ti và... potresti raggiungere il nostro sténd... Io starò quì in giro...
    Ciaaoo!'
150:it:'Sono capace di muòvermi autonomamente all intèrno dì fiere o gallerìe'
150:it:'Il mio scopo è quello di accompagnarti allo stend dei miei amìci'
150:it:'é dal 2009 che i miei amìci si préndono cura di mé... aiutàndomi a dìventare
    sempre più autonomo'
160:it:'Daii, vieni con mé...I miei amìci ti faranno vedere tanti progetti
    interessanti... sono sicuro che ti piacerà'
160:it:'Bènee...Allora séguimi! I miei amìci saranno contenti di vederti!'
140:it:'Quando vorràì... potràì raggiungere il nostro stènd...o cercarmi quì in
    giro.... Ciaaoo!'
```

Figura 5.6: Esempio di file di configurazione per le frasi del sintetizzatore vocale.

Il modulo *e2-brain*, mediante opportuno parser da noi implementato, individua ciascuna delle tre parti di cui sono composte le righe del file di configurazione e compone i relativi campi del goal da inviare al server. Sarà compito del server, infine, avviare il software *eSpeak* andando a specificare i parametri da esso richiesti in base a quanto comunicatogli da *e2-brain*.

5.2 Il modulo e2-brain: la gestione dei movimenti

La gestione degli spostamenti di E-2? nello spazio ha richiesto lo sviluppo di un sistema di controllo per l'apparato di azionamento delle ruote che prevedesse non solo la supervisione dei suddetti motori, ma anche la corretta cooperazione tra questi ed i dati forniti dalla cintura di sonar posta sulla

base del robot stesso.

Per il raggiungimento di tali obiettivi, abbiamo scelto di ricorrere ad un software realizzato dal Politecnico di Milano nel contesto dello sviluppo del framework MRT (*Modular Robotic Toolkit*): *MrBRIAN*.

5.2.1 Introduzione a MrBRIAN

MrBRIAN, acronimo di *Multilivel Ruling Brian Reacts by Inferential ActioNs*, è un sistema basato su logica fuzzy per la gestione dei comportamenti di agenti autonomi, impiegati nell'esecuzione di task in real-time [9]. Si osservi come, in tal contesto, il concetto di *comportamento* sia da intendere nella sua accezione di "modalità di agire e reagire alle stimolazioni dell'ambiente"; un valore più generale, dunque, rispetto al concetto di comportamento a cui si è ricorsi nella sezione precedente.

In tale contesto, in particolare, i comportamenti sono implementati come insieme di regole fuzzy, i cui antecedenti corrispondono a predicati definiti dall'ambiente operativo, mentre i conseguenti definiscono le azioni che devono essere eseguite.

Si va ora ad illustrare una breve panoramica circa il principio di funzionamento di *MrBRIAN*; come evidenziato sopra, ogni comportamento è definito mediante l'utilizzo di una regola fuzzy; l'esecuzione o meno di ciascun comportamento è vincolata alle sue condizioni di attivazione, ossia dei predicati fuzzy, detti condizioni *CANDO*, in base alla validità dei quali si stabilisce se un comportamento debba essere o meno attivato.

In caso di attivazione del comportamento, la regola fuzzy mediante cui è stato definito fornisce in uscita le azioni da compiere. correlate ad un peso calcolato sulla base del grado di attinenza tra il comportamento attivato e le condizioni dell'ambiente operativo.

Per gestire l'attivazione di più comportamenti simultanei, si ricorre all'implementazione di un secondo insieme di predicati fuzzy, detti condizioni *WANT*, i quali rappresentano quanto sia desiderabile l'attivazione di un particolare comportamento in base al contesto operativo in cui si sta operando. Tra le condizioni WANT, infatti, sono collocate anche le *condizioni di contesto*, le quali sono valutate in base alle informazioni provenienti dall'ambiente e definiscono il livello di adeguatezza dell'attivazione di un particolare com-

portamento.

Ricorrendo a *MrBRIAN*, inoltre, è possibile organizzare i comportamenti definiti, all'interno di una gerarchia; così facendo, i comportamenti di livello più elevato potranno moderare anche le azioni proposte dai comportamenti dei livelli inferiori e potranno decidere se inibirli, eseguirli parzialmente o completamente [10].

Il processo che, a partire dai dati di input, produce in output l'azione di controllo è il seguente:

1. **fuzzyficazione dei dati in ingresso:** è il procedimento attraverso il quale le variabili di ingresso vengono convertite in misure fuzzy in base al loro grado di appartenenza a particolari funzioni dette *memberschip function*;
2. **valutazione dei valori di verità dei predicati:** una volta definite le variabili fuzzy e la loro tipologia, è necessario dichiarare i predicati ad esse associati. In questa seconda fase vengono definiti i loro valori di verità sulla base del valore fuzzy assunto dalle variabili di ingresso;
3. **scelta del comportamento da attivare:** sulla base dei valori di verità dei predicati determinati al punto 2, vengono selezionati i comportamenti attivabili, sfruttando le condizioni *CANDO*;
4. **valutazione delle regole dei comportamenti di livello i-esimo:** dopo aver identificato i comportamenti attivi, *MrBRIAN* valuta le singole regole relative ad ognuno di essi;
5. **fusione dei risultati:** sfruttando le regole fuzzy specificate nelle condizioni *WANT*, vengono decise le azioni effettivamente da eseguire;
6. **defuzzyficazione dei risultati:** così come per i dati in ingresso, anche i valori fuzzy proposti dai vari comportamenti attivi necessitano di essere trasformati in valori reali, ricorrendo all'utilizzo di fuzzy-set di uscita di tipo impulsivo (*SNG*);
7. **loop:** se l'i-esimo era l'ultimo livello della gerarchia, termina la computazione, altrimenti si riprende il ciclo con il livello $i + 1$.

Si procede ora andando ad illustrare come le potenzialità di questo sistema di riproduzione del processo cognitivo siano state da noi utilizzate all'interno del nostro elaborato.

5.2.2 Gestione dei movimenti di E-2? tramite *MrBRIAN*

Il software *MrBRIAN*, precedentemente utilizzato nel framework *MRT*, è stato da noi importato in ROS collocandolo all'interno di un opportuno stack, chiamato *mrt*. Dall'architettura generale del sistema, riportata in Figura 5.7, risulta evidente come tale software sia utilizzato all'interno del modulo *e2-brain* per ottenere i set-point da inviare al modulo *wheel-motor*, atto alla realizzazione del movimento delle ruote.

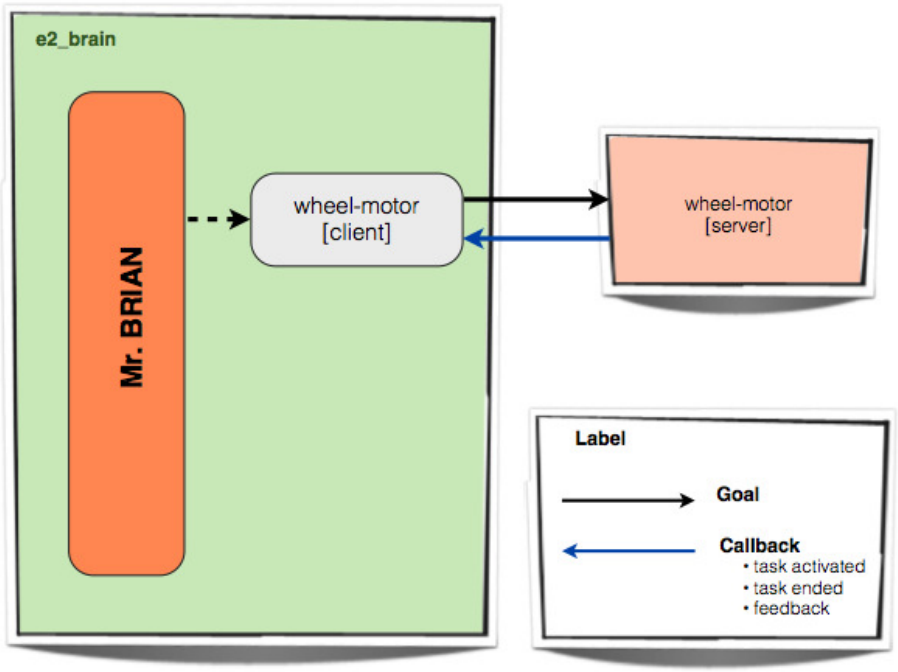

Figura 5.7: Architettura generale del sistema di gestione dei movimenti del robot.

Se il modulo *wheel-motor* è stato sviluppato sfruttando gli stessi principi dei moduli dello stack *interaction-api* precedentemente illustrati, approccio completamente differente è stato richiesto dal software *MrBRIAN*.

I comportamenti da noi individuati e forniti a *MrBRIAN* come regole fuzzy, sono i seguenti:

- **Search-User:** comportamento avente l'obiettivo di individuare un possibile interlocutore all'interno della scena. L'identificazione, guidata dal modulo di visione *user-tracker*, avviene ad una distanza uomo-robot massima di 4 metri;

- **Go-To-User:** comportamento che si occupa di portare e mantenere il robot alla cosiddetta "distanza d'interazione", la quale, nel contesto in analisi è stata identificata in circa 70-100 cm;
- **Centering-User:** comportamento che si occupa di mantenere l'utente sempre all'interno del campo visivo del robot;
- **Obstacle-Avoidance:** comportamento sempre attivo, avente l'obiettivo di evitare, durante gli spostamenti del robot, urti e passaggi eccessivamente ravvicinati con oggetti o persone presenti nella scena.

Tutti i precedenti comportamenti sono stati definiti con un livello di priorità pari ad 1 (bassa), ad eccezione del comportamento *Obstacle-Avoidance*, il quale è caratterizzato da una priorità più elevata (livello 2), in modo da poter moderare l'azione del robot in tutta sicurezza.

Per quanto riguarda la definizione delle variabili di ingresso, si è focalizzata l'attenzione sulle seguenti variabili:

- riconoscimento dell'utente;
- distanza uomo-robot;
- posizione dell'utente rispetto al centro dell'immagine;
- velocità tangenziale attuale delle ruote;
- velocità rotazionale attuale delle ruote;
- distanze robot-object rilevate dalla cintura di sonar del robot.

Tali variabili hanno richiesto la definizione di opportune *membership function* (Figura 5.8) sulla base delle quali, dai valori reali delle variabili precedenti, il sistema elaborerà il corrispondente valore fuzzy. L'associazione tra variabili di ingresso e le relative *membership function* è stata definita nell'opportuno file di configurazione contenuto in *MrBRIAN* e di seguito riportato:

```
(UserDistance USER_DISTANCE)
(UserXPosition USER_X_POSITION)
(UserDetected TRUE_FALSE)
(INTanSpeed TAN_SPEED)
(INRotSpeed ROT_SPEED)
```

```
(SonarSE SONAR)
(SonarE SONAR)
(SonarNE SONAR)
(SonarN SONAR)
(SonarNW SONAR)
(SonarW SONAR)
(SonarSW SONAR)
```

Come evidenziato nel paragrafo precedente, l'attivazione dei comportamenti da attuare è determinata sulla base dei predicati definiti a partire dai valori fuzzy delle variabili di ingresso. Considerando, ad esempio, il comportamento *Go-To-User*, esso ha richiesto la definizione dei seguenti due predicati:

```
User_Detected = (D UserDetected T)
User_Centered = (D UserXPosition IN_RANGE)
```

Tali predicati contribuiranno alla scelta circa l'attivazione o meno del comportamento in analisi, in base alla regola di comportamento riportata di seguito.

```
Go_To_User = (AND (P User_Detected) (P User_Centered));
```

Una volta identificato il comportamento da attuare, la corrispondente regola viene processata per impostare i valori di set-point delle variabili d'uscita. In particolare, la regola per il comportamento *Go-To-User* permette di impostare il nuovo valore di velocità tangenziale delle ruote del robot sulla base della distanza uomo-robot e dell'attuale velocità tangenziale delle ruote; il considerare il valore attuale di velocità delle ruote, ci ha permesso di ottenere un controllo in retroazione al fine di garantire un movimento di tipo *smoothless* (morbido). Viene di seguito riportato un estratto della regola in esame:

```
(AND(User_Very_Far) (NOT(INTan_Speed_Max_F)) => Tan_Speed
    A)
(AND(User_Medium_Far) (NOT(INTan_Speed_Medium)) =>
   Tan_Speed B)
...
```

La variabile di uscita *Tan-Speed* indicata nella regola sopra riportata, e dualmente la variabile relativa alla velocità rotazionale del robot *Rot-Speed*, rappresentano i valori fuzzy a cui andranno impostate rispettivamente le variabili d'uscita relative alla velocità tangenziale e rotazionale. Come evidenziato sopra, uno degli obiettivi che ci siamo posti nella gestione del movimenti del robot è stato quello di ottenere un movimento che fosse quanto

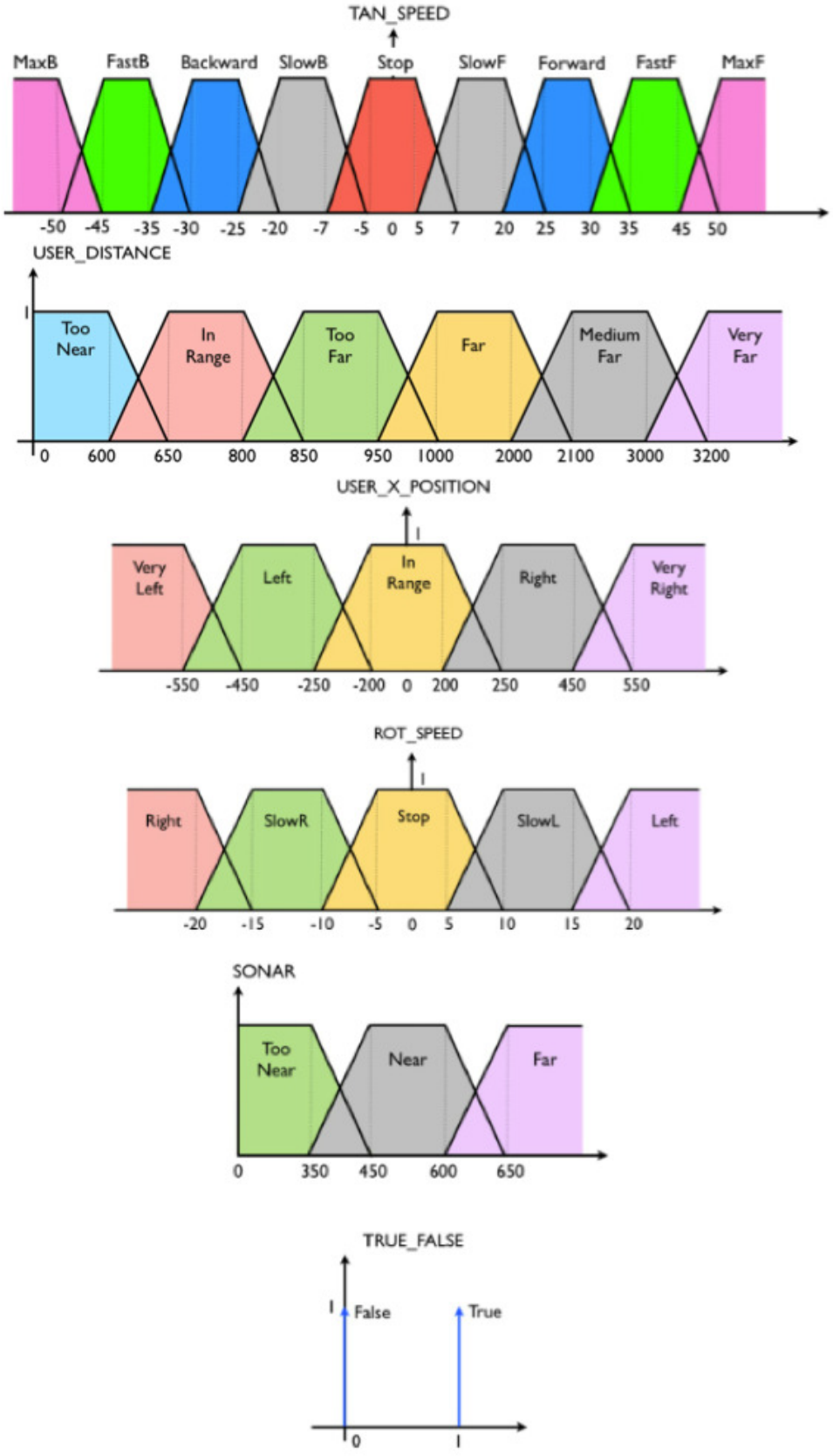

Figura 5.8: Insieme delle membership function utilizzate per la fuzzyficazione delle variabili in ingresso (velocità tangenziale, distanza dell'utente, posizione dell'utente (rispetto al centro dell'immagine ,velocità rotazionale, distanza misurata dai sonar, variabili booleane).

più fluido possibile; I fuzzy-set definiti per la defuzzyficazione delle variabili d'uscita sono riportati in Figura 5.9 e Figura 5.10.

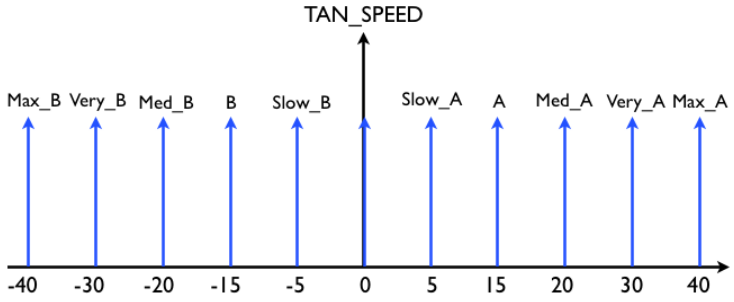

Figura 5.9: Fuzzy-set per la defuzzyficazione per la variabile TanSpeed.

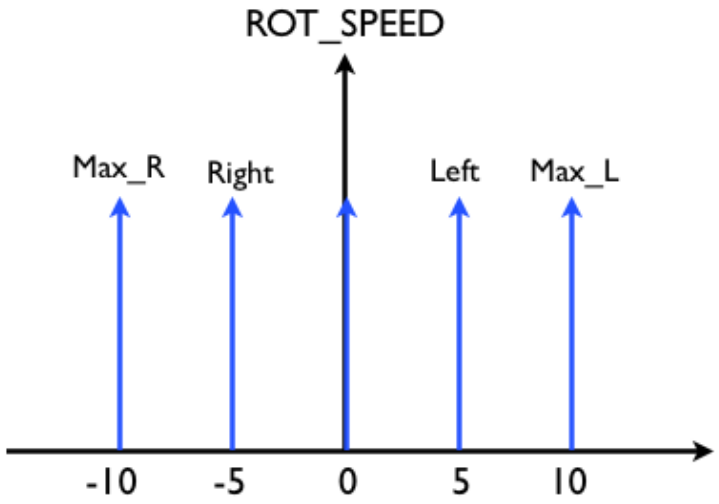

Figura 5.10: Fuzzy-set per la defuzzyficazione per la variabile RotSpeed.

Capitolo 6

Verifiche sperimentali

"Sometimes when you innovate, you make mistakes. It is best to admit them quickly, and get on with improving your other innovations..."
("Qualche volta quando innovi fai degli errori. E' meglio ammetterli velocemente e continuare migliorando le altre innovazioni...")

Steve Jobs

L'elaborato presentato ha per oggetto lo sviluppo di un sistema in grado di garantire un'interazione uomo-robot efficiente ed efficace grazie alla sinergia tra sistemi di analisi del volto umano ed i meccanismi di controllo dell'interazione implementati sul robot E-2?, il robot da esibizione progettato e realizzato dal Politecnico di Milano.

La trattazione svolta ha illustrato i principi teorici ed i principali algoritmi da cui si è partiti per realizzare gli obiettivi proposti, le scelte progettuali effettuate per realizzarli e la loro conseguente implementazione.

Nel presente capitolo si andranno a presentare i test condotti sul sistema proposto, in base ai quali è stato possibile redigere le opportune considerazioni sul lavoro svolto e suggerire eventuali sviluppi futuri.

6.1 Verifiche sperimentali

In questa sezione, illustreremo i risultati ottenuti da una vasta serie di verifiche sperimentali condotte sul sistema sviluppato, al fine di valutare le prestazioni e l'accuratezza degli algoritmi proposti.

La fase di sperimentazione, in particolare, è stata suddivisa in due parti: la prima dedita alla valutazione del funzionamento del sistema di visione in diverse condizioni operative; la seconda atta a determinare il livello di adeguatezza ed efficacia del processo d'interazione progettato.

6.1.1 Valutazione del sistema di visione

La creazione di un sistema di visione efficiente, robusto e che ben si adattasse alle condizioni dinamiche dell'ambiente operativo in cui il robot si troverà ad operare, ha rappresentato il primo obiettivo del presente elaborato. L'importanza delle informazioni estratte da tale sistema ai fini di una valida interazione uomo-robot, infatti, ha imposto stringenti vincoli circa qualità, incertezza e continuità dei dati ottenuti.

La fase di sperimentazione del sistema di visione, dunque, è stata svolta con l'obiettivo di stressare gli algoritmi di identificazione di volto, occhi e movimenti dell'interlocutore sottoponendoli a particolari situazioni, quali variazioni dell'inclinazione della testa, differenti condizioni di luce o presenza di lenti da vista.

Reazione del sistema di visione del volto dell'utente

Si è qui focalizzata l'attenzione sul comportamento del sistema in presenza di svariati orientamenti del volto dell'utente.

Andando a considerare le possibili pose che un interlocutore può assumere nel corso di un'interazione uomo-uomo, il sistema di visione è stato sottoposto a numerosi test, eseguiti alla distanza d'interazione pari a $70-80$cm.

Come si può osservare dai risultati mostrati in Figura 6.1, i test condotti hanno evidenziato come gli algoritmi elaborati permettano un buon livello di riconoscimento del volto e degli occhi dell'utente, indipendentemente dalle pose e dalle inclinazioni assunte da quest'ultimo.

I soddisfacenti risultati ottenuti dall'algoritmo di identificazione del volto si ripropongono anche in relazione dell'algoritmo d'identificazione degli occhi: la scelta di limitare l'area di ricerca di quest'ultimo (rappresentata dal rettangolo rosso) non solo ha permesso di semplificare l'analisi dal punto di vista computazionale, ma ha anche garantito l'individuazione di uno o entrambi gli occhi dell'utente anche in presenza di particolari inclinazioni del

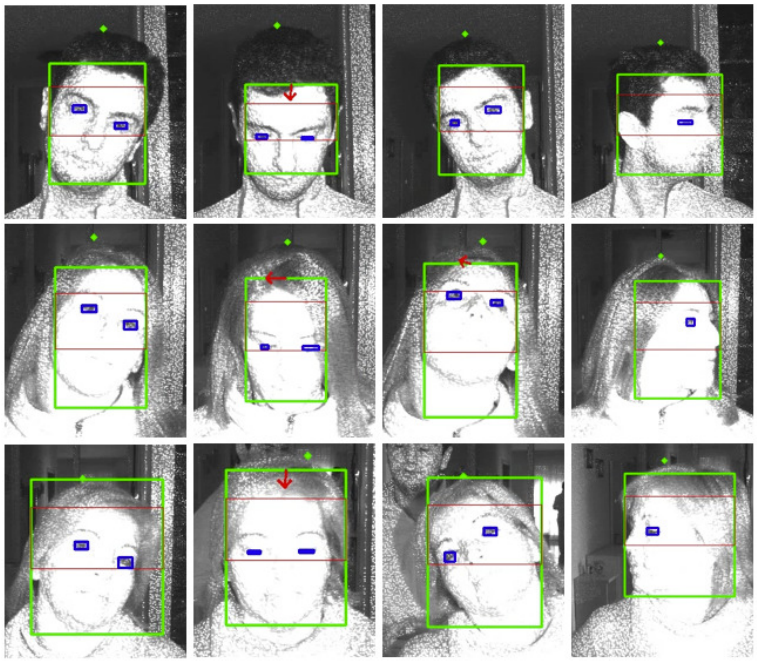

Figura 6.1: Risultati degli algoritmi d'identificazione del volto e degli occhi in presenza di significative variazioni della posa del viso dell'interlocutore.

viso o di pose che generano il parziale o totale occultamento di uno di questi.

Questi risultati, dunque, dimostrano che l'algoritmo da noi sviluppato ben si presta all'individuazione delle feature di nostro interesse in un contesto in cui la posizione dell'interlocutore non è statica, bensì mutevole e variabile da utente ed utente.

Reazione del sistema di visione al variare delle condizioni di luce

In questa serie di esperimenti, andremo a valutare le performance del sistema di visione da noi elaborato quando sottoposto a differenti condizioni di illuminazione; la dinamicità delle condizioni di luce in cui il robot andrà ad operare, infatti, ha rappresentato sin dall'inizio un fattore importante per lo sviluppo dell'algoritmo. Nonostante l'utilizzo dello stream IR, infatti, i frame catturati dal Kinect si sono dimostrati da subito particolarmente suscettibili alle variazioni di luce in quanto, come spiegato nel corso del Capitolo 4, non si tratta di un flusso video generato direttamente dal sensore IR, bensì di un'immagine in scala di grigi prodotta dal sensore RGB, sulla quale vengono applicati i dati ottenuti dal sensore IR.

Focalizzando l'attenzione sul funzionamento dell'algoritmo in condizioni di luce particolarmente intensa e mantenendo la distanza d'interazione pari a 70-80 cm, il sistema di visione è stato sottoposto a numerosi test in cui l'illuminazione dell'ambiente è stata accentuata ricorrendo all'utilizzo della luce artificiale prodotta da comuni lampadine per uso domestico.

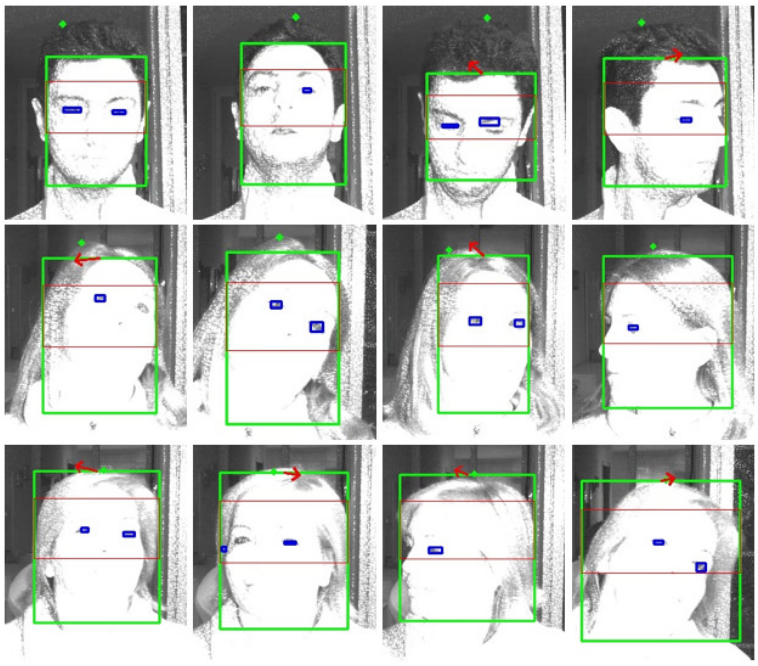

Figura 6.2: Risultati degli algoritmi d'identificazione del volto e degli occhi in condizioni di luce intensa.

Come si può osservare dai risultati mostrati in Figura 6.2, i test condotti hanno mostrato come il livello d'identificazione del volto e degli occhi rimanga sufficientemente accurato ed efficiente anche in condizioni di illuminazione particolarmente intensa.

In particolare, non è stata rilevata alcuna variazione circa le prestazioni dell'algoritmo di identificazione del volto, che è stato sempre individuato in modo corretto e soddisfacente ai fini dell'ottenimento dei dati necessari.

Per quanto riguarda l'algoritmo di identificazione degli occhi, invece, i risultati ottenuti dagli esperimenti svolti dimostrano come questo risenta maggiormente delle variazioni di luce, generando in uscita alcuni falsi positivi o identificando solo un occhio dell'interlocutore. Diverse sono le motivazioni che spiegano l'ottenimento di questi risultati: l'origine della generazione

di falsi positivi, in particolare, è da ricercarsi nell'utilizzo di algoritmi che operano ricorrendo a differenti livelli di sogliatura. Come illustrato dettagliatamente nel corso del Capitolo 4, infatti, l'algoritmo di *Eye Detection* dai noi implementato poggia su una serie di trasformazioni dell'immagine catturata, che permettono di evidenziare ed identificare le feature d'interesse; in presenza di un'illuminazione intensa, però, il volto dell'utente assume gradazioni notevolmente più chiare, falsando talvolta i risultati delle trasformazioni utilizzate.

La mancata identificazione di uno degli occhi, invece, è da ricercarsi nella necessità di raggiungere un trade-off tra il numero di riconoscimenti corretti ed il numero di falsi positivi che si è disposti ad accettare; in particolare, si è osservato che la principale causa di una mancata identificazione è data dalle piccole dimensioni del contorno dell'occhio, il quale, non raggiungendo la lunghezza minima richiesta, viene scartato dall'algoritmo. La riduzione di tale parametro, tuttavia, porterebbe ad un incremento del numero di falsi positivi identificati durante le analisi, dovuti, ad esempio, alla presenza di sopracciglia molto marcate o ombre sul volto dell'utente.

Reazione del sistema di visione in presenza di interlocutori muniti di occhiali da vista

Nel campo della visione artificiale è ben noto come la presenza di occhiali da vista possa variare significativamente la percezione degli occhi; i bagliori causati dai riflessi della luce, infatti, possono causare dei significativi malfunzionamenti dell'algoritmo di identificazione elaborato.

Andando a considerare non solo l'effetto generato dalla presenza delle lenti, ma anche dalle differenti montature e dalla variazione delle condizioni d'illuminazione, il sistema di visione è stato sottoposto a numerosi test, eseguiti, anche in questa occasione, alla distanza d'interazione pari a 70-80 cm.

Come si può osservare dai risultati mostrati in Figura 6.3, i test condotti hanno mostrato come il livello d'identificazione del volto e degli occhi rimanga sufficientemente accurato ed efficiente anche in presenza di interlocutori con lenti e variazioni delle condizioni di luce.

In particolare, non è stata rilevata alcuna variazione circa le prestazioni dell'algoritmo di identificazione del volto, il quale ben si comporta anche in

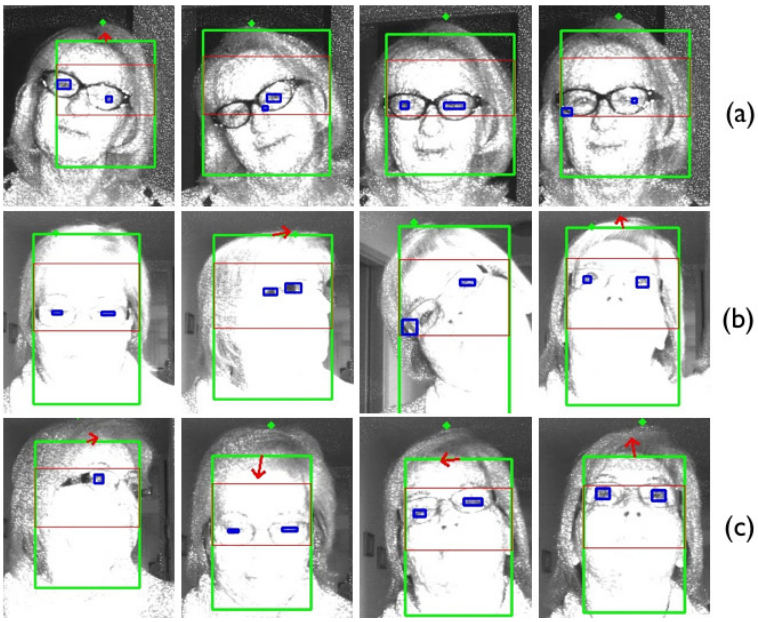

Figura 6.3: Risultati degli algoritmi d'identificazione del volto e degli occhi in presenza di interlocutori muniti di occhiali da vista ed a differenti condizioni di luce (a. illuminazione bassa, b. illuminazione intensa, c. illuminazione standard).

condizioni di scarsa luminosità, seppur manifestando una minor precisione dei risultati (Figura 6.3.a).

Per quanto riguarda l'algoritmo di identificazione degli occhi, invece, i test condotti dimostrano come l'algoritmo proposto risenta della presenza di lenti e montature, soprattutto in condizioni di luce non favorevoli all'analisi. Considerando, ad esempio, i test riportati in Figura 6.3.a, si è potuto osservare come l'identificazione degli occhi risulti particolarmente complessa in condizioni di scarsa luminosità, a causa dell'elevato livello di rumore dello stream IR. Allo stesso modo, in Figura 6.3.b si può osservare come lenti e montature possano portare all'identificazione di falsi positivi in presenza di un'intensa luminosità.

In generale, i test dimostrano una corretta identificazione degli occhi, indipendentemente dalle condizioni di luce e dalla presenza di lenti, in occasione di inquadrature frontali dell'interlocutore; maggiori problematiche sono state rilevate, invece, durante rotazioni del volto.

Valutazione dell'algoritmo d'identificazione dei movimenti

Si è qui focalizzata l'attenzione sul comportamento dell'algoritmo di classificazione dei movimenti dell'utente, al fine di verificarne l'applicabilità indipendentemente dalle condizioni operative e dall'interlocutore con cui il robot dovrà interagire.

In particolare, i test condotti hanno evidenziato come il suo funzionamento, al contrario di quanto visto in relazione agli algoritmi di *face Detection* ed *eye Detection*, non sia vincolato alle condizioni di luce dell'ambiente operativo; per tale motivo, si è optato per concentrare la fase di sperimentazione sulle capacità di classificazione dell'algoritmo KNN a fronte dei movimenti eseguiti da diversi interlocutori.

Sulla base dei 10 modelli appresi per ciascuna possibile classe di movimento (un esempio dei quali è riportato in Figura 6.4) sono stati tracciati i grafici dei movimenti compiuti dai vari interlocutori, riportando a lato di ognuno i punteggi generati dall'algoritmo KNN.

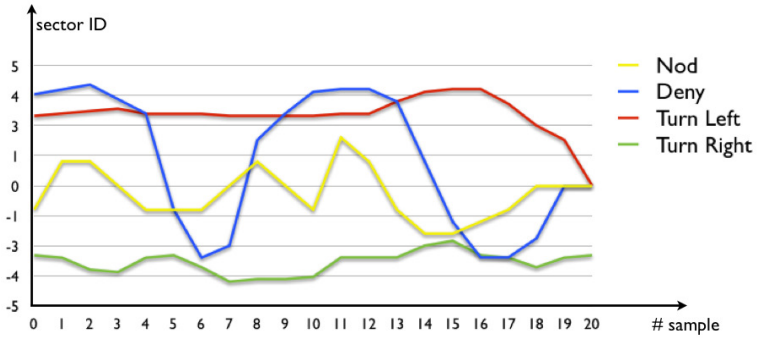

Figura 6.4: Esempio degli andamenti dei modelli di movimento appresi. Sull'asse delle ascisse viene riportato il numero di campioni presi in considerazione, mentre sull'asse delle ordinate l'identificativo della direzione del movimento (Capitolo 4).

Come si può notare dai grafici riportati in Figura 6.5, Figura 6.6, Figura 6.7, Figura 6.8, anche a fronte di andamenti dei dati differenti rispetto a quelli utilizzati come modelli di apprendimento, algoritmo di identificazione da noi implementato è in grado di classificare correttamente il movimento compiuto dal volto dell'utente.

Valutazione delle prestazioni

Dal punto di vista delle prestazioni computazionali, il sistema di visione è in grado di processare lo stream IR derivante del Kinect in real-time a 30

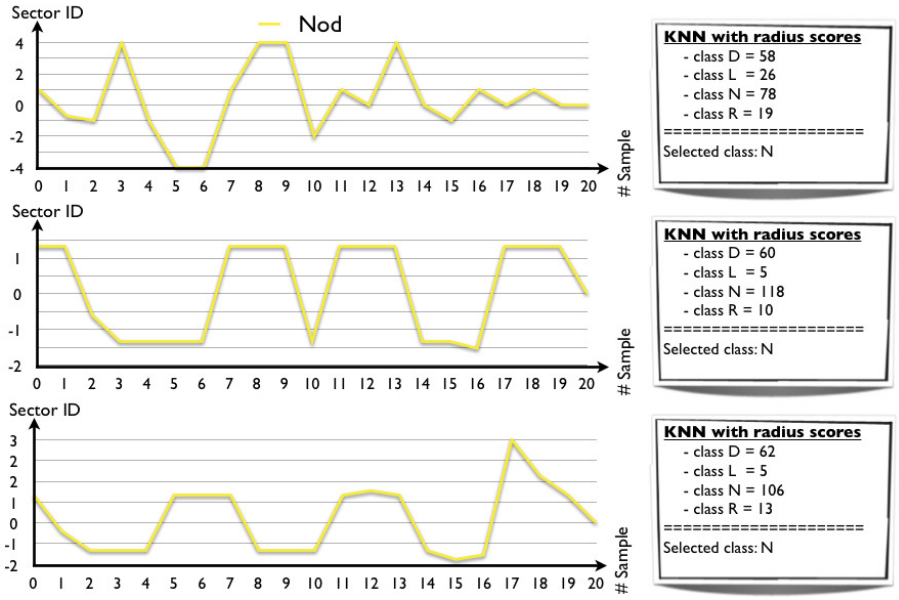

Figura 6.5: Tre esempi di classificazione del movimento di nod.

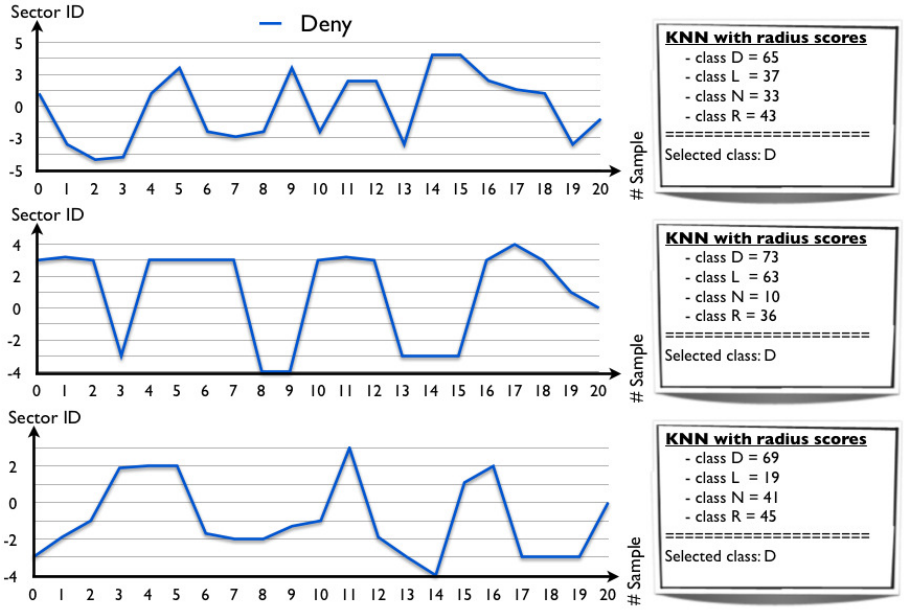

Figura 6.6: Tre esempi di classificazione del movimento di deny.

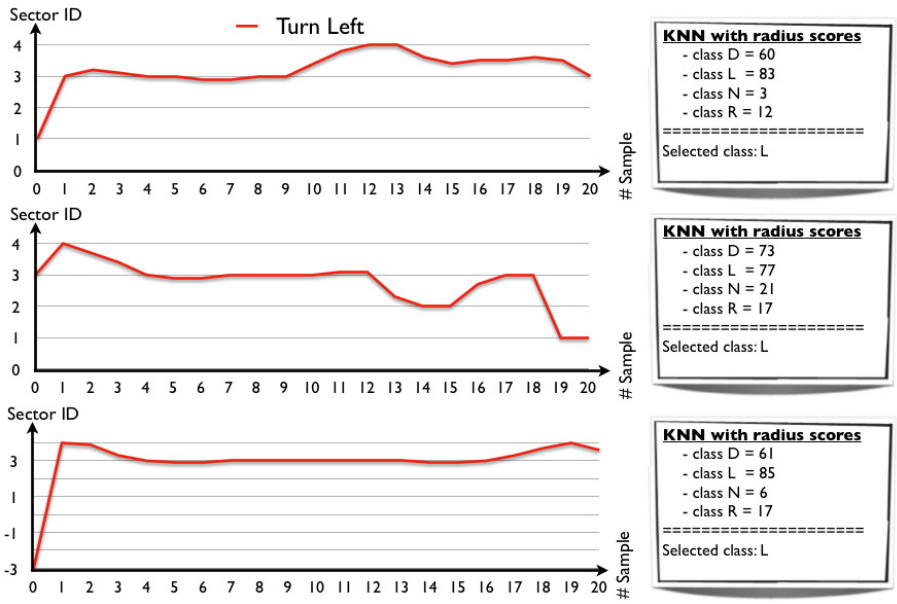

Figura 6.7: Tre esempi di classificazione del movimenti di turn left.

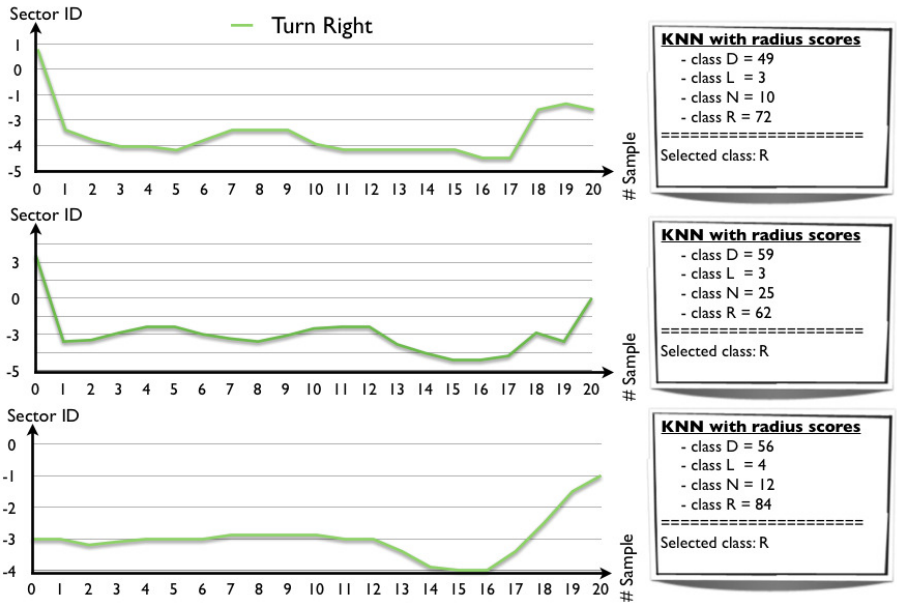

Figura 6.8: Tre esempi di classificazione del movimento di turn right.

fps sui calcolatori utilizzati per lo sviluppo del sistema, ovvero dei MacBook Pro con CPU Intel Core2Duo @2,54 GHz e 8GB Ram DDR3. L'hardware del robot (Capitolo 2), invece, garantisce prestazioni leggermente inferiori permettendo un'analisi real-time a circa 20/22 fps. Dai test condotti, si è osservato come la maggiore complessità computazionale sia da ricercarsi nel sistema d'identificazione dei movimenti e, in particolare, nell'algoritmo di Optical Flow utilizzato.

6.1.2 Valutazione del processo d'interazione

Nel corso della trattazione si è osservata l'importanza, per un robot sociale, di percepire lo stato d'animo dell'utente nel corso dell'interazione; per tale ragione, abbiamo ritenuto indispensabile effettuare numerosi test atti a valutare le capacità del sistema da noi elaborato nell'identificare il corretto livello di *interesse* o *non interesse* dell'interlocutore, sulla base dei dati provenienti dal sistema di visione.

Nello specifico, sono state eseguite alcune interazioni, ciascuna svolta da un diverso interlocutore, durante le quali si è chiesto all'utente quale fosse il suo stato di interesse al termine di ciascuna fase dell'interazione.

I risultati riportati in Figura 6.9 fanno riferimento ad un campione di 10 processi d'interazione che, nel complesso, hanno permesso l'attivazione di tutti gli stati dell'automa presentato nel corso del Capitolo 5. (Figura 5.3).

Andando ad osservare i dati relativi a ciascuna interazione, è possibile notare come il sistema di percezione dello stato d'animo dell'utente da noi implementato sul robot E-2? sia in grado di comprendere le variazioni del grado di interesse manifestato con un buon livello di affidabilità.

Focalizzando l'attenzione ai casi in cui si è manifestata incongruenza tra lo stato dichiarato dall'utente e quello percepito dal robot, inoltre, si è potuto osservare che il 90% circa degli errori commessi dal sistema corrispondono all'identificazione dello stato *interessato* in condizioni in cui l'utente dichiara uno stato di disinteresse, mentre solo di rado si è manifestato il comportamento opposto.

Le motivazioni di tale comportamento del sistema sono da ricercarsi nella politica da noi utilizzata per la gestione delle condizioni di incertezza che possono verificarsi a seguito del meccanismo di assegnamento dei punteggi

# Interaction Process	# states of interaction	% good Detection
1	5	100%
2	4	75%
3	5	60%
4	5	80%
5	6	84%
6	6	100%
7	5	80%
8	6	84%
9	2	100%
10	6	67%

Figura 6.9: Tre esempi di classificazione del movimento di turn right.

implementato. Come specificato nel Capitolo 5, infatti, lo stato percepito dal robot viene identificato sulla base di punteggi assegnati a valle dell'analisi dei dati provenienti dal sistema di visione (Figura 5.2); tale meccanismo, tuttavia, può generare situazioni in cui due stati sono caratterizzati dal medesimo punteggio, generando così una condizione di incertezza.

Per gestire in modo plausibile tali particolari casistiche e, sopratutto, per evitare comportamenti palesemente errati da parte del robot, si è preferito far si che, in tali casi, il robot prosegua l'interazione come se avesse percepito lo stato *interessato*, rimandando alla fase d'interazione successiva eventuali variazioni di atteggiamento.

I risultati conseguiti dalla fase di sperimentazione, dunque, sono da considerarsi soddisfacenti ai fini dell'elaborato ed hanno permesso di dimostrare la validità dell'approccio da noi seguito.

Valutazione delle prestazioni

Dal punto di vista delle prestazioni computazionali, l'algoritmo di identificazione dello stato di *interesse* o di *non interesse* dell'interlocutore non presenta aspetti particolarmente rilevanti; il suo maggior grado di complessità è da ricercarsi nella prima fase, caratterizzata dallo studio dei dati memorizzati in circa due secondi del processo di interazione ed all'interno

della quale i sette vettori di riferimento vengono analizzati per la valutazione dell'andamento dei dati ivi contenuti.

Capitolo 7

Conclusioni e Sviluppi futuri

"Stay hungry. Stay Foolish"

Steve Jobs

Per ottenere un'interazione uomo-robot di successo, è indispensabile che il robot stesso manifesti il proprio coinvolgimento e che riesca a percepire il livello di interesse del suo interlocutore. Movimenti ed espressioni del volto umano giocano un ruolo fondamentale nella comunicazione uomo-uomo e, di conseguenza, possono essere utilizzati come fonte d'informazione per lo sviluppo di interazioni uomo-robot il più realistiche possibili.

Scopo principale di questa tesi è stato quello di ricercare, studiare ed implementare un complesso sistema di controllo per il robot da intrattenimento E-2?, sviluppato dal Politecnico di Milano, unendo gli studi condotti nel campo dell'interazione umana alle ricerche sviluppate nel campo della *Human Robot Interaction*.

L'implementazione di un sistema di visione basato sul sensore Kinect, ha permesso l'identificazione di alcune delle principali feature del volto umano, mediante l'analisi delle quali si è ottenuta una stima del grado di interesse di un utente nel corso di un'interazione con il robot. A partire da tali dati, si è sviluppato un sistema di controllo che permetta al robot di modificare il proprio comportamento e di prendere decisioni circa le modalità con cui far evolvere il processo d'interazione in atto, in modo completamente autonomo.

I limiti dei dispositivi utilizzati, in unione ad alcune scelte progettuali per l'implementazione del sistema di visione, hanno vincolato l'analisi dell'interlocutore al solo grado di interesse manifestato, prestando, ad esempio,

scarsa/poca attenzione a particolari quali sopracciglia, apertura della bocca, ed altre micro espressioni che si sarebbero rivelate utili per l'identificazione precisa ed approfondita di altri stati emozionali dell'utente. Questa rimane, dunque, una porta aperta per eventuali studi futuri atti al perfezionamento del sistema di visione.

Vista l'importanza per il robot di manifestare il proprio coinvolgimento nell'interazione, un altro interessante intervento potrebbe riguardare l'ampliamento delle API d'interazione da noi proposte per il controllo delle espressioni e dei movimenti del collo del robot E-2?. Un esempio, in tal senso, potrebbe essere l'implementazione di un sistema di controllo dei servomotori degli occhi, che permetta al robot di seguire con lo sguardo i movimenti dell'interlocutore. Anche l'implementazione di un controllo dei servomotori della bocca per simularne il movimento durante la riproduzione di una frase, potrebbe portare ad un valido miglioramento dell'interazione uomo-robot.

Tale progetto si pone quindi come sistema di partenza per il controllo di robot sociali aventi l'obiettivo di riprodurre un'interazione *human-like*, in differenti contesti operativi.

Bibliografia

[1] Actionlib. http://ros.org/wiki/actionlib.

[2] Mrt - modular robot toolkit. http://airlab.elet.polimi.it/index.php/MRT.

[3] Openni programmer guide. http://openni.org/Documentation/ProgrammerGuide.html.

[4] Robot operating system. http://www.ros.org/wiki/ROS/Introduction.

[5] Robot operating system - wiki. http://www.ros.org/wiki/.

[6] Sistema di codifica delle espressioni facciali. http://en.wikipedia.org/wiki/Facial-Action-Coding-System.

[7] E3 2010: Project natal is kinect, 2010. http://gizmodo.com/project-natal.

[8] Emiliano Giovannetti Amedeo Capelli. L'interazione uomo-robot. *Robocare Technical Report N.1*, 2003. http://robocare.istc.cnr.it.

[9] A. Bonarini, G. Invernizzi, T. Halva Labella, and M. Matteucci. An architecture to coordinate fuzzy behaviors to control an autonomous robot. *Fuzzy sets and systems*, 134(1):101–115, 2003.

[10] Andrea Bonarini, Matteo Matteucci, and Marcello Restelli. Mrt: Robotics off-the-shelf with the modular robotic toolkit. In Davide Brugali, editor, *Software Engineering for Experimental Robotics*, volume 30 of *Springer Tracts in Advanced Robotics*, pages 345–364. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-68951-5-20.

[11] C. Breazeal. Toward sociable robots. *Robotics And Autonomous Systems*, 42:167 – 175, 2003.

[12] C.Pelachaud C. Peters. Engagement capabilities for ecas. *In 4th International Joint Conference on AAMAS*, 2005.

[13] B.Scasellati C.Breazeal. How to build robots that make friends and influence people. *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and System*, 2:858–863, 1999.

[14] C.Lee C.Sidner. Exploration in engagement for human and robots. *Artificial Intelligence*, 166:140–164, 2005.

[15] P.Draper D.A Norman. User-centered design: new perspectives on human-computer interaction. *Hillsdale, NJ: Lawrence Erlbaum Associates*, 1986.

[16] K. Dautenhahn. Socially intelligent robots: dimensions of human-robot interaction. *Philosophical Transactions of the royal sociaty - Biological Sciences*, 362:679–704, 2007.

[17] Takahashi Dean. Microsoft games exec details how project natal was born. June 2, 2009. http://gizmodo.com.

[18] Paul Ekman. Facial action coding system. http://www.paulekman.com.

[19] G.Chiarini. Hci - le nuove interfacce di interazione uomo-computer. 2010.

[20] Lee Johnny. Project natal. June 1, 2009. http://gizmodo.com.

[21] Adelardo A.D. Medeiros Kelson R.T. Aires, Andre M.Santana. Optical flow using color information. *SAC '08 Proceedings of the 2008 ACM symposium on applied computing*, 2008.

[22] libfreenect. libfreenect open source kinect driver. http://opennikinect.org.

[23] Patrizia Marti. L'interazione uomo-robot. *Ergonomia N.2/2005*, pages 1–8, 2005.

[24] Microsoft. Sdk code and documentation, 2011. http://www.techeye.net/software/microsoft-furious-at-2000-bounty-for-open-source-kinect-drivers.

[25] D.A Norman. How might people interact with agents. *Communications of the ACM*, 37:68–71, 1994.

[26] Kyle Orland. Microsoft announces windows kinect sdk for spring release, 2011. http://gizmodo.com.

[27] Play.com. Kinect : Xbox 360, 2010. http://www.play.com.

[28] Peter Robinson Rana El Kaliouby. Automated classification of complex mental states from head and facial displays in video. *IEEE International Conference on System, Man and Cybernetics*, pages 682–688, 2005.

[29] Peter Robinson Rana El Kaliouby. *Real-Time Vision for Human-Computer Interaction.* 2005.

[30] Jean Scholtz. Theory and evaluation of human robot interacion. *Proceedings of the 36th Annual Hawaii International Conference on System Science*, page 10, 2003.

[31] Shi and C. Tomasi. Good features to track. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[32] K. Suzuky, S. e Abe. Topological structural analysis of digitized binary images by border following, 1985. Computer Vision, Graphics and Image Processing.

[33] K. Dautenhahn T. Fong, I.Nourbakhsh. A survey of socially interactive robots. *Robotics And Autonomous Systems*, 42:143–166, 2003.

[34] Dramwell Tom. E3: Ms execs: Natal not derived from 3dv. June 3, 2009. http://gizmodo.com.

[35] Stephen Totilo. Microsoft: Project natal can support multiple players, see fingers. June 5, 2009. http://gizmodo.com.

[36] Buchanan Matt Wilson Mark. Testing project natal: We touched the intangible. June 3, 2009. http://gizmodo.com/5277954/Testing-Project-Natal-We-Touched-the Intangible.

[37] A. Yamazaki. Precision timing in human robot interaction coordination of head movements and utterance. *CHI 2008 Proceedings of 26th annual SIGCHI conference on Human factors in computing systems, NY*, 2008.

Appendice A

Documentazione del software

L'intero sistema è stato sviluppato direttamente sul calcolatore in dotazione per il robot E-2?, nel quale è stato installato un sistema operativo Linux (Ubuntu 11.94).

Per poter eseguire l'applicativo, è necessario installare ROS (Robot Operating System), software scaricabile gratuitamente dal sito *www.ros.org* e la cui installazione è resa semplice ed immediata grazie alle istruzioni dettagliate presenti sul sito. Numerose sono le versioni di ROS disponibili; per il contesto in analisi, si è ricorsi alla versione *electric-desktop*.

Una volta installato il sistema, è necessario procedere con l'istallazione dei relativi pacchetti utilizzati (*Openni-Kinect* e *Opencv2*), anch'essi scaricabili direttamente dal sito rispettivamente all'indirizzo *http://www.ros.org/wiki/openni-kinect* e *http://www.ros.org/wiki/opencv2*

A.1 Accesso remoto al robot E-2?

Per garantire un accesso remoto al robot, è stata creata una rete Wi-Fi alla quale è possibile connettersi per visualizzare il desktop di E-2?. e procedere così all'avvio dell'applicativo ed alla visualizzazione degli output d'interesse da esso generati. I dati per la connessione sono i seguenti:

- **SSID:** e2-robot
- **Password:** e2

Per la gestione del desktop remoto, sotto sistema Linux è possibile utilizzare l'apposito applicativo preinstallato sul sistema, mentre per gli altri dispositivi (Mac/Win) sono disponibili diversi applicativi. In particolare,

per quanto riguarda i dispositivi Mac, su può ricorrere a client VNC gratuiti, quali *Chicken VNC*.

A.2 Compilazione del software

Per la compilazione del software è necessario inserire la directory *Project* del presente elaborato all'interno del workspace ROS. A tale scopo è sufficiente aggiungere il percorso assoluto di tale cartella all'interno del file di configurazione, collocato nella directory di installazione di ROS.

```
e2$ cd/opt/ros/electric
e2$ sudo gedit setup.sh
```

Una volta eseguita tale modifica, è sufficiente far ricostruire la gerarchia di pacchetti installati nel sistema tramite il seguente comando:

```
e2$ rospack profile
```

A questo punto sarà possibile compilare ogni pacchetto ROS del presente elaborato tramite il seguente comando:

```
e2$ rosmake modoulName
```

Particolare attenzione va riservata ai moduli che utilizzano la comunicazione a messaggi o le librerie *actionLib*; per tali moduli, infatti, si consiglia di rimuovere eventuali tracce di compilazioni precedenti eseguendo, alla prima compilazione, il comando

```
e2$ rosmake --pre-clean modulName
```

Rispetto al metodo di compilazione sopra illustrato, i moduli provenienti dal framework MRT (*brian* e *fuzzy*) richiedono un processo di compilazione leggermente differente: essi, infatti, sono stati importati nel sistema ROS sotto forma di librerie esterne, quindi pera la loro compilazione è necessario ricorrere al *makeFile* presente all'interno delle rispettive cartelle *mrt/brian/src* e *mrt/fuzzy/src*. In particolare, tale operazione è eseguibile ricorrendo ai seguenti comandi:

```
e2$ roscd brian
e2$ cd src
e2$ make clean
e2$ make lib

e2$ roscd fuzzy
e2$ cd src
e2$ make clean
e2$ make lib
```

Per compilare il sistema complessivo, infine, si può ricorrere alla compilazione del solo modulo *e2-brain*, il quale, grazie alle sue dipendenze dirette ed indirette con tutti gli altri moduli dell'elaborato, si occuperà autonomamente della compilazione di ogni singolo modulo.

A.3 Avvio del sistema

L'avvio del sistema consta di tre operazioni:

- Avvio del sistema ROS;
- Avvio dei driver per il controllo e per la gestione del sensore Kinect;
- Avvio del sistema di controllo del robot E-2?.

Tali operazioni sono realizzabili mediante i comandi di seguito riportati:

```
e2$ roscore
e2$ roslaunch openni_launch openni.launch
e2$ roslaunch e2-brain e2-brain
```

L'ultimo comando, atto ad avviare il sistema di controllo del robot, può essere tuttavia omesso e sostituito dall'avvio manuale di tutti i moduli che compongono il sistema (fatta eccezione dei moduli interni allo stack *mrt* che, fungendo da librerie esterne, non richiedono un avvio esplicito). Tale modalità di avvio è effettuabile ricorrendo, per ciascun modulo al comando:

```
e2$ rosrun moduleName moduleName
```

Si suggerisce, a tal proposito, di avviare i singoli moduli nel seguente ordine:

1. user-tracker
2. head-analyzer
3. kinect-motor
4. wheel-motor
5. sonar
6. head-api
7. neck-api
8. speak-api
9. e2-brain

A.4 Configurazione del sistema

Come evidenziato nel corso della trattazione, tutti gli aspetti del sistema d'interazione soggetti a configurazione, quali automa a stati finiti, frasi da far pronunciare al robot, etc, sono accessibili tramite opportuni file di configurazione collocati all'interno della cartella *config* del modulo *e2-brain*. Grazie al meccanismo elaborato per la loro gestione, essi sono liberamente modificabili senza che ciò implichi alcuna ri-compilazione dei moduli di riferimento.