POLITECNICO DI MILANO
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica

E2GoHome
Sistema di controllo per un robot da esibizione

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

Relatore: Prof. Andrea Bonarini

Tesi di Laurea di:
Lorenzo Ripani, matricola 739364

Anno Accademico 2013-2014

alla mia famiglia ...

Sommario

L'interesse sempre crescente nei campi della robotica e dell'intelligenza artificiale, ha portato a notevoli sviluppi nelle interazioni tra uomo e robot (Human Robot Interaction - *HRI*), che sempre più frequentemente riscontriamo nella vita di tutti i giorni. Inoltre la possibilità di disporre di sensori performanti a basso costo, ha incentivato la costruzione di robot capaci di percepire l'ambiente ad un livello sempre più profondo, incrementando la capacità di rispondere prontamente ad eventi esterni che si presentano operando in un mondo dinamico, sottoposto a repentini cambiamenti.

Lo scopo di questa tesi è stato quello di realizzare un sistema di controllo per un robot mobile da esibizione, che permetta al robot di interagire in piena autonomia con le persone e gli garantisca il supporto necessario per muoversi liberamente nell'ambiente senza la supervisione umana.

Il documento illustra il sistema di controllo e il processo di interazione ottenuto attraverso l'utilizzo di sensori montati sul robot, e mostrando l'importanza della HRI quale chiave di sviluppo nei robot di nuova generazione.

Ringraziamenti

Desidero ricordare tutti coloro che mi hanno aiutato nella stesura della tesi con suggerimenti, critiche ed osservazioni: a loro va tutta la mia gratitudine.

Ringrazio anzitutto il professor Andrea Bonarini, che in questi mesi mi ha seguito ed aiutato nel fronteggiare ogni problema che si presentava, proponendomi sempre nuove sfide che mi hanno aiutato a crescere ed a confrontarmi con problemi reali permettendomi di avere una visione più completa del mondo della robotica.

Ringraziando inoltre tutto il personale e i ragazzi dell'AIR-Lab, in particolare Martino Migliavacca, che mi ha aiutato nella risoluzione dei misteriosi problemi della base del robot. Un ringraziamento particolare va anche a Stefano Cappelli che mi ha coadiuvato durante lo sviluppo del codice del collo dell'agente, passando intere serate all'interno dei laboratori.

Infine ringrazio tutte le persone che mi sono state accanto durante questo periodo sopportando le mie follie, e la mia famiglia che in questi anni mi ha sempre sostenuto aiutandomi a raggiungere questo traguardo. A tutti voi un grazie di cuore.

Indice

Capitolo 1

Introduzione

"... you just can't differentiate between a robot and the very best of humans."

Isaac Asimov, I, Robot

Le interazioni uomo-robot sono da sempre oggetto di discussione, sia nel mondo accademico, sia in quello letterario e nel corso degli anni si sono lentamente inserite nella vita di tutti i giorni, spesso modificando il nostro rapporto con il quotidiano. Nonostante i robot siano stati tipicamente utilizzati in attività statiche e ripetitive, oggi sono sempre più coinvolti in compiti complessi e meno strutturati, in particolare nella collaborazione e nell'interazione con l'uomo. In questo ambito si colloca il presente elaborato, che vuole mostrare un sistema di controllo autonomo per un robot da esibizione, tale *E-2?*; verranno descritte le azioni che la macchina è in grado di compiere e come questa riesca ad interagire con le persone durante la navigazione nell'ambiente. Lo scopo di *E-2?* e' quello di avvicinare persone in un ambiente fieristico, fornire loro informazione rispetto a un espositore, rilevarne l'eventuale interesse, nel qual caso deve offrirsi di accompagnare la persona allo stand dell'espositore in questione.

Per il conseguimento di tali obbiettivi si è resa necessaria l'analisi di numerose informazioni provenienti dai sensori distribuiti sul corpo del robot, quali telecamere IR, sonar ed encoder. Un importante contributo per l'ottenimento di questi dati è stato fornito dal dispositivo Kinect, che in questo elaborato viene utilizzato come sensore di navigazione e di localizzazione e per l'analisi condotta sul volto delle persone

con le quali il robot stabilisce un'interazione.

Il robot è stato oggetto di diverse modifiche durante lo sviluppo del progetto, che hanno portato alla sostituzione della precedente base mobile e ad una ristrutturazione del software che permettesse a tutte le componenti di dialogare fra di loro per il corretto funzionamento del sistema. Successivamente è stato sviluppato un sistema di navigazione che consente al robot di spostarsi autonomamente nell'ambiente, allo scopo di cercare persone con cui iniziare un processo di interazione. Infine lo sviluppo si è concentrato sull'analisi del volto delle persone, permettendo al robot di memorizzarne il volto, in modo da poterle riconoscere nel corso del tempo.

L'elaborato si articola in sei capitoli, che descrivono nel dettaglio le componenti sviluppate e come queste collaborino al raggiungimento dei propri obiettivi e dell'obiettivo globale. I capitoli sono strutturati come elencato di seguito.

Capitolo 2 *Stato dell'arte*: vengono presentati i recenti sviluppi ottenuti in merito a tematiche di HRI , riconoscimento facciale e motion planning, descrivendo il mondo della robotica e gli argomenti che hanno influenzato lo sviluppo dell'elaborato.

Capitolo 3 *Architettura del sistema*: nella prima parte, vengono esposti i principali componenti hardware del robot E-2?, utilizzati per il raggiungimento degli obiettivi del progetto. Nelle seconda parte, invece, viene presentato l'ambiente ROS, utilizzato come framework di sviluppo e l'architettura del software realizzato.

Capitolo 4 *Il sistema di navigazione*: vengono qui presentati i moduli sviluppati con la relativa documentazione, in merito al sistema di navigazione del robot. In particolare vengono illustrati il processo di localizzazione ed il sistema di motion planning che consente al robot di muoversi autonomamente nell'ambiente.

Capitolo 5 *Il sistema di visione*: qui viene descritto il processo di rilevamento dei volti tramite il sensore Kinect, focalizzando l'attenzione su come viene effettuata la memorizzazione e la successiva identificazione

delle persone con cui il robot intrattiene un'interazione.

Capitolo 6 *Processo di interazione*: vengono descritte le componenti che collaborano al processo di interazione che il robot stabilisce con una persona, descrivendo dettagliatamente i nodi coinvolti.

Capitolo 7 *Verifiche Sperimentali*: vengono presentati i risultati ottenuti nei test effettuati, allo scopo di valutare il processo di navigazione e di visione del robot. Vengono infine mostrati possibili accorgimenti attraverso i quali sarebbe possibile aumentare il livello di coinvolgimento della persona durante l'interazione.

Da ultimo nell'**appendice A** viene riportato il manuale dell'utente per l'utilizzo del software e del simulatore adoperato nelle varie fasi di sviluppo.

Capitolo 2

Stato dell'arte

"Imparare senza pensare è fatica perduta; pensare senza imparare è pericoloso."

Confucio, Dialoghi

Gli sviluppi tecnologici degli ultimi decenni unitamente ai progressi nel campo dell'intelligenza artificiale e dell'elaborazione del linguaggio naturale, hanno portato ad un sostanziale sviluppo nel campo della robotica, con applicazioni sempre più orientate verso l'interazione uomo-robot.

Uno dei principali sviluppi ha visto la graduale introduzione dei robot all'interno della società, con impatto significativo nei vari campi di applicazione. Le nuove generazioni di robot, classificati come robot sociali, nascono con l'obiettivo di offrire alle persone un'esperienza piacevole, basata su un'interazione amichevole ed informativa. Questo livello di comunicazione, tuttavia, richiede un gran numero di abilità di cui deve disporre la macchina. Per questa ragione, lo studio di sistemi robotici in grado di percepire l'ambiente in modo complesso e capaci di interagire con esso, è uno dei temi più attuali nei principali laboratori di robotica nel mondo, soprattutto riferito all'interazione uomo-robot e allo sviluppo di interfacce basate sull'elaborazione del linguaggio.

Utilizzare un robot in un contesto dinamico abitato da esseri umani, impone precisi requisiti riguardanti la percezione sensoriale, la mobilità e la destrezza, nonché la capacità di pianificare compiti, prendere decisioni ed effettuare ragionamenti. In questo capitolo verranno discus-

si i principali argomenti che hanno influenzato lo sviluppo di questo progetto.

2.1 Robotica

La robotica ha radici culturali assai profonde. Nel corso dei secoli l'uomo ha costantemente cercato di individuare dei suoi sostituti che risultassero in grado di emulare il suo comportamento nelle molteplici occasioni di interazione con l'ambiente circostante.

La robotica ha stimolato la fantasia di molti scrittori che con la loro immaginazione hanno guidato lo sviluppo tecnologico degli ultimi decenni. Il termine robotica si riferisce allo studio e all'uso dei robot, termine diffusosi tramite i libri dello scrittore Isaac Asimov [8] (Fig. 2.1).

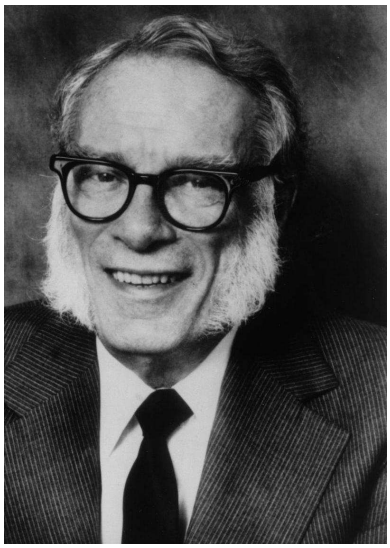


Figura 2.1: Isaac Asimov

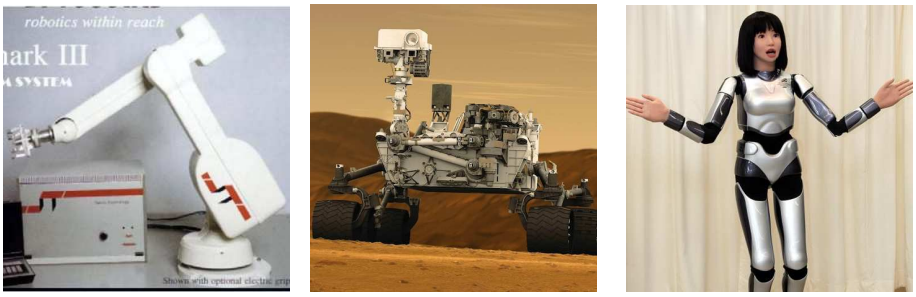
A lui si deve l'idea delle leggi della robotica, riportate qui di seguito, e la visione di un mondo in cui i robot diventano parte integrante della società e nel quale la robotica costituisce una potente industria.

- **Legge zero**: *Un robot non può causare danno all'umanità, o, a causa della sua inazione, permettere che l'umanità venga danneggiata.*
- **Prima legge**: *Un robot non può causare danno ad un essere umano, o permettere che un essere umano venga danneggiato.*
- **Seconda Legge**: *Un robot deve obbedire ad un ordine impartito da un essere umano.*
- **Terza legge**: *Un robot deve proteggere se stesso.*

La visione che Asimov enuncia nei suoi libri, non si discosta di molto da quanto effettivamente accaduto negli ultimi anni, che hanno visto la lenta ma graduale introduzione dei robot nella vita di tutti i giorni, 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 macchine ispirate alle caratteristiche biologiche e comportamentali di organismi animali a quella di robot sociali ispirati alle modalità di relazione e comunicazione degli esseri umani.

Ripercorrendo la storia della robotica, si nota come ogni nuova generazione sviluppata tenda ad aumentare il proprio livello di autonomia, assumendo comportamenti sempre più inclini alla socialità e alla collaborazione. Inizialmente, i primi sviluppi erano incentrati su applicazioni rigorosamente orientate nel campo industriale, con la nascita dei primi manipolatori programmabili (Fig. 2.2a). In seguito, gli sforzi si sono concentrati verso il rapporto macchina-ambiente, garantendo un migliore percezione dell'ambiente di lavoro; tutto questo si è tradotto nella creazione di agenti mobili in grado di muoversi nell'ambiente circostante e capaci di interagire con gli oggetti che li circondano. Esempi evidenti di robot mobili si hanno nel campo dell'esplorazione spaziale (Fig. 2.2b), ove il robot deve far fronte ad eventi imprevedibili e quindi necessita di un elevato grado di autonomia. Infine gli ultimi sviluppi hanno portato ad una nuova generazione di robot con sembianze antropomorfe, creando di fatto le prime forme di androidi: ne sono esempi



(a) Manipolatore (b) Mars Rover (c) HRP-4C

Figura 2.2: Alcune tipologie di Robot

Actroid e HRP-4C (Fig. 2.2c).

HRP-4C è un androide con un volto femminile creato dall'istituto nazionale Giapponese per la scienza e la tecnologia (AIST) e mostrato in pubblico a Tokio il 16 marzo 2009 [21]. Il robot è dotato di capacità di riconoscimento e sintesi vocale ed è in grado di compiere movimenti attraverso piccoli passi e mostrare differenti stati emotivi tramite un sistema di espressioni facciali.

Questo esempio mostra il tentativo di creare agenti sempre più complessi, in grado di poter interagire con la società che li circonda. Questi sviluppi hanno portato alla nascita dei cosiddetti robot sociali, descritti nel paragrafo successivo.

2.2 Robot Sociali

Secondo la definizione di Fong et al. [15], i robot sociali rappresentano agenti inseriti in un gruppo eterogeneo: siano essi robot o esseri umani. Sono in grado di riconoscersi, iniziare una interazione sociale e sono in grado di comunicare ed apprendere gli uni dagli altri.

Breazeal [10] fornisce una classificazione dei robot sociali, basata sia sulla capacità del robot di supportare il modello sociale che gli è stato attribuito, sia sulla complessità dello scenario d'interazione che il robot è in grado di gestire. Le categorie identificate sono:

- **Suggestivi** (socially evocative): 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): robot circondati da un ambiente sociale, in grado di percepirlo e di reagire ad esso. I robot di questa categoria sono capaci di distinguere, nel contesto dell'ambiente, gli esseri umani da qualsiasi altro oggetto;

- **Socievoli** (sociable): robot che, di loro 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): robot che manifestano caratteristiche dell'interazione sociale tipica dell'uomo, basandosi su modelli della cognizione umana e delle competenze sociali.

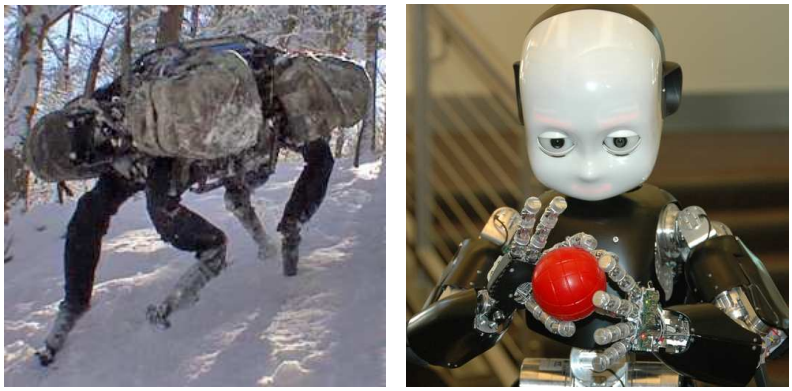
Il termine robot sociali interattivi (Socially Interactive Robots) identifica tutti quegli agenti per i quali l'interazione sociale rappresenta un ruolo fondamentale, distinguendo così tali macchine da quelle caratterizzate da un'interazione uomo-robot convenzionale, come i robot utilizzati per gli scenari di teleoperazione.

Un'altra caratteristica importante dei robot interattivi sociali, è il loro design. Molti robot infatti, assumono sembianze sempre più antropomorfe per soddisfare le aspettative di comunicazione tipiche degli esseri umani. In questo modo la comunicazione tra un robot ed un essere umano sarà più piacevole per una persona, se questa si trova a suo agio. Per questo motivo spesso vengono inseriti occhi e labbra meccanici capaci di simulare movimenti, e dare l'impressione di una comunicazione con un con agente intenzionale. Per quanto attiene al design, questi robot possono essere ulteriormente classificati in due modi. Un primo approccio consiste in un design biologico, che tenta riprodurre comportamenti o azioni tipiche del mondo animale [7] (Fig. 2.3a, 2.3b, 2.4a); un'altra possibilità, definita come design funzionale, si pone l'obiettivo di costruire un robot che appaia esteriormente socialmente intelligente, nonostante il suo design interno non sia corredato

(a) Cebrennus rechenbergi *(b) Spider Robot*

Figura 2.3: Robot ispirato al movimento del ragno Cebrennus rechenbergi

da una base rigorosamente scientifica [24] (Fig. 2.4b).



(a) Big Dog *(b) iCub*

Figura 2.4: Design per alcuni robot

In conclusione, al fine di decidere quali abilità sociali e fisiche siano utili per un robot, è importante attuare un'analisi dettagliata di diversi aspetti, quali il dominio applicativo, la natura e la frequenza con cui il robot entrerà in contatto e interagirà con l'uomo.

2.3 Modalità di interazione

La comunicazione tra un robot ed un essere umano può avvenire sia attraverso i dispositivi classici dell'Interazione Uomo-Computer (tastiera, mouse, monitor, ecc.) oppure utilizzando modalità interattive più naturali (human-friendly), tipiche dell'interazione uomo-uomo, in grado di coinvolgere tutti i sensi umani e i canali di comunicazione, come il linguaggio, la visione, la gestualità ed il tatto.

Dal lavoro Cappelli et al. [13], si possono estrapolare sei categorie principali di modalità d'interazione del robot:

- **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. Con l'aumentare del livello di complessità nei robot e la possibilità di eseguire compiti specifici, il linguaggio naturale appare un'alternativa più che desiderabile rispetto alla selezione di un comando tramite tastiera o attraverso la visualizzazione su di uno schermo.

- **Gesti**: il riconoscimento dei gesti umani è un'area di ricerca in continuo sviluppo e diversi studi si sono interessati al ruolo della gestualità 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 [12]. Un esempio ricorrente in letteratura, in merito alla capacità di un robot di produrre espressioni facciali è Kismet (Fig. 2.5a), poi sostituito da Leonardo, un robot costruito presso l'Artificial Intelligence Laboratory del MIT [11]. 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. Altri esempi sono Paro [22] (Fig. 2.6), un robot di compagnia per persone anziane, anche questo dotato di pelliccia ed una ampia gamma di movimenti riproducibili.

(a) Kismet *(b) Leonardo*

Figura 2.5: Robot costruiti presso l'Artificial Intelligence Laboratory del MIT

- **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 il punto focale ove la persona sta guardando e a cosa sta prestando attenzione, sarà in grado di capire, ad esempio, se questa persona si sta rivolgendo al robot o ad altri;

- **Prossemica e cinesica**: modalità di comunicazione più sofisticate sono la prossemica e la cinesica: la prima riguarda 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 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 opportuna-

mente interpretata, risulta essere una preziosa fonte aggiuntiva di informazione;

- **Aptica**: tutto quanto attiene al senso del tattile, può essere classificato con il termine aptica, definito come l'acquisizione dell'informazione e la 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 è Leonardo (Fig. 2.5b). Questo piccolo robot 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 2.6: Paro - Robot terapeutico per anziani e bambini

Come descritto da questi studi, 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, è necessario sfruttare le tecniche a disposizione per far si che il robot manifesti il più possibile la sua intenzionalità: l'uomo deve poter credere che il robot abbia opinioni, desideri e intenzioni.

2.4 Visione Artificiale

Uno degli obiettivi primari della Computer Vision consiste nell'identificazione di oggetti e corpi all'interno di immagini digitali bidimensionali o 3D. Un sistema di visione artificiale è costituito dall'integrazione di componenti ottiche, elettroniche e meccaniche che permettono di acquisire, registrare ed elaborare immagini sia nello spettro della luce visibile che al di fuori di essa. Il risultato dell'elaborazione è il riconoscimento di determinate caratteristiche dell'immagine per varie finalità di controllo, classificazione, selezione e via dicendo.

Un Sistema di Visione si compone di diverse componenti:

- telecamere e ottiche;
- sistema di illuminazione;
- il sistema di acquisizione e di elaborazione dell'immagine;
- le interfacce uomo-macchina;
- le interfacce con l'ambiente esterno;

Ad esempio, in un sistema di ispezione visiva per controllo di qualità, le parti da ispezionare vengono posizionate - spesso attraverso sistemi di movimentazione automatica - di fronte a una o più telecamere ed illuminati in modo appropriato, in modo cioè da evidenziare il più possibile i difetti da individuare. Il sistema ottico forma un'immagine sul sensore della telecamera che produce un segnale elettrico in uscita. Questo segnale verrà digitalizzato e memorizzato. L'immagine, catturata e resa in questo modo comprensibile da un calcolatore, potrà quindi essere elaborata con un apposito software che comprende particolari algoritmi di calcolo ed analisi, in grado di individuare le caratteristiche dell'immagine e amplificarne alcuni aspetti - ad esempio contorni, spigoli, forme, strutture - allo scopo di eseguire i controlli e le verifiche per i quali il sistema è stato concepito. Sulla base dei risultati dell'elaborazione il sistema prenderà decisioni in merito alla destinazione dell'oggetto, ad esempio smistarlo fra i buoni o scartarlo e fornirà le informazioni opportune al resto del sistema produttivo.

2.4.1 Campi Applicativi

Un problema classico nella visione artificiale è quello di determinare se l'immagine contiene o no determinati oggetti (Object recognition). Il problema può essere risolto efficacemente e senza difficoltà per oggetti specifici in situazioni specifiche per esempio il riconoscimento di specifici oggetti geometrici come poliedri, riconoscimento di volti o caratteri scritti a mano. In letteratura troviamo differenti varietà del problema:

- **Recognition**: uno o più oggetti prespecificati o memorizzati possono essere ricondotti a classi generiche usalmente insieme alla loro posizione 2D o 3D nella scena;
- **Identification**: viene individuata un'istanza specifica di una classe. Es. Identificazione di un volto, impronta digitale o veicolo specifico;
- **Detection**: L'immagine e scandita fino all'individuazione di una condizione specifica. Es. Individuazione di possibili cellule anormali o tessuti nelle immagini mediche o di difetti in parti meccaniche o tessuti.

Altro compito tipico è la ricostruzione della scena: date 2 o più immagini 2D si tenta di ricostruire un modello 3D della scena. Nel caso più semplice si parla di un set di singoli punti 3D. Casi più complessi tentano di generare superfici 3D complete. Generalmente è importante trovare la matrice fondamentale che rappresenta i punti comuni provenienti da immagini differenti.

Una parte significativa dell'intelligenza artificiale si occupa di gestire sistemi interfacciati a robot o macchine che si muovono nello spazio o compiono movimenti. Questa tipologia di processi implica spesso l'acquisizione di informazioni fornite da un sistema di visione artificiale che occupa il ruolo di sensore visivo. Esistono sistemi in grado di dirigere manipolatori, robot antropomorfi o carrelli in ambienti industriali non noti a priori. Ad esempio robot di carico e scarico che devono individuare la posizione esatta di oggetti diversi e posizionarli su pallet o in contenitori o ad un sistema di movimentazione intelligente, in grado di muoversi in uno stabilimento dove circolano persone, altri mezzi di movimentazione e spesso le aree sono occupate da merci.

Un'area di applicazione emergente è quella dei veicoli autonomi come sommergibili, veicoli terrestri su ruote o cingolati o veicoli volanti. Un sistema di visione artificiale può sia supportare un pilota di questi veicoli in varie situazioni, sia occuparsi dell'intera navigazione, nel caso di veicoli fully-autonomous. In questo caso, è importante saper riconoscere gli ostacoli e riuscire a produrre una mappa della zona circostante per capire dove sia possibile muoversi. Esempi in questa area sono warning-system nelle automobili, sistemi per l'atterraggio automatico degli aerei o sistemi per la guida automatica di autovetture. Quest'ultima tecnologia anche se studiata e prodotta non ha ancora raggiunto i costi e l'affidabilita' sufficienti per essere lanciata sul mercato. Le applicazioni militari sono probabilmente una delle più grandi aree che sfrutta i benefici della visione artificiale anche se solo una piccola parte del lavoro svolto in questo ambiente viene reso pubblico. Esempi sono i sistema di guida di missili e droni sugli obiettivi.

Una delle maggiori e promettenti aree è quella medica. Quest'area è caratterizzata dall'estrazione di informazioni dall'immagine con l'intento di effettuare la diagnosi di un paziente. Tipicamente l'immagine è acquisita attraverso microscopia, raggi X, angiografia e tomografia. Esempi di informazioni che possono essere dedotte dalle immagini sono la presenza di tumori, arteriosclerosi o altre disfunzioni maligne.

2.4.2 Organizzazione di un sistema di visione

L'organizzazione del sistema di visione dipende molto dal campo di applicazione. Alcuni sistemi sono singole applicazioni che risolvono uno specifico problema di rilevamento o una misurazione, mentre altri costituiscono sottosistemi più ampi, contenendo anche attuatori meccanici di movimento, database, interfacce uomo-macchina. Un'altra caratteristica relativa all'implementazione dipende anche dal tipo di operazioni da compiere, ad esempio se sono funzionalità prestabilite o se è necessario adattare i modelli di elaborazione durante le operazioni.

Tuttavia, si possono stabilire alcuni processi che si possono trovare nella maggior parte dei sistemi visivi:

1. **Acquisizione Immagine** Una immagine è prodotta da uno o più sensori, che possono presentarsi sotto diverse forme: dispositivi sensibili alla luce, sensori di profondità, radar, camere ad ultrasuoni. A seconda dei sensori l'immagine risultante potrà essere

2D o 3D, o una sequenza di immagini. I valori dei pixel tipicamente corrispondono all'intensità luminosa percepita in una o più bande spettrali (scale di grigio o a colori), ma possono essere associati anche a misurazioni fisiche come profondità o assorbimento di onde soniche o elettromagnetiche;

2. **Pre-processing** Prima di poter applicare ogni tipo di analisi all'immagine per estrapolare delle informazioni, è necessario processare l'immagine per determinare se soddisfa i vincoli imposti dall'analisi da effettuare. Ad esempio alcuni metodi proposti da questa tesi richiedono che le immagini siano su scala di grigio e quindi devono essere processate per rispettare tali vincoli.

3. **Estrazione Feature** Le caratteristiche dell'immagine vengono estrapolate a vari livelli di complessità per determinare contorni, o estrapolare pattern utili all'identificazione di un oggetto.

4. **Rilevamento** Ad un certo punto, nel processo viene presa una decisione riguardo a quali punti o porzioni dell'immagine sono rilevanti o per effettuare ulteriori elaborazioni;

5. **Elaborazioni Alto Livello** In questa fase i dati di analisi sono tipicamente ridotti e ci si trova nelle condizioni di poter effettuare le analisi finali o altro a seconda del tipo di sistema che si è costruito.

Capitolo 3

Architettura del sistema

"I am definitely not a fucking toaster..."

Jim Chaseley, Z14

In questa sezione viene mostrata l'architettura di E-2? (Fig. 3.1), un robot sviluppato all'interno del laboratorio di intelligenza artificiale e robotica del Politecnico di Milano, che è stato oggetto di studio nel corso di questo progetto.

Il robot si compone di tre parti principali: una base mobile, un collo articolato ed una testa meccanica in grado di mostrare differenti espressioni facciali. La testa (Fig. 3.1a), oltre a contenere al suo interno tutta la meccanica ed i servomotori per il controllo di occhi, sopracciglia e bocca, sostiene il sensore Kinect, utilizzato per il sistema di visione artificiale e come sensore di navigazione. Il collo è stato progettato per garantire da un lato, una buona rigidità di tutta la struttura e dall'altro, per avere un buon numero di movimenti riproducibili. Esso è costituito da 5 servomotori che conferiscono i gradi di libertà necessari per compiere anche movimenti particolarmente complessi. La base del robot, infine, contiene al suo interno, tre coppie di motori elettrici per il controllo delle ruote che ne permettono il movimento, un mini-PC e sei batterie da 12 V che alimentano tutte le componenti del robot.

Tutto il controllo viene effettuato sul calcolatore del robot che utilizza il framework ROS per supportare e facilitare lo scambio e la comunicazione dei vari sottosistemi della macchina.

(a) Volto E-2? *(b) E-2?*

Figura 3.1: E-2? - robot sviluppato presso AIRLab - Politecnico di Milano

3.1 Componenti Hardware

Di seguito vengono presentate le principali componenti hardware del robot E-2? con particolare riferimento a quelle che hanno avuto un ruolo decisivo nello sviluppo dell'elaborato.

3.1.1 Base Mobile

La struttura su cui poggia il robot è una base mobile omni-direzionale (Fig. 3.2), composta da tre bracci in alluminio, che sostengono la piattaforma in ABS (Acrilonitrile-Butadiene-Stirene) ove sono alloggiate le schede ed il calcolatore del robot. La struttura fisica della base è stata realizzata all'interno dell'AIRLab, con elementi meccanici della Item, ed i motori e i sottosistemi di trasmissione sono stati progettati cercando di minimizzare il numero di elementi meccanici [6]. Ogni braccio della struttura si compone dei seguenti elementi:

- Una combinazione motore, trasmissione ed encoder della Maxon, che con i 70 Watt erogati fornisce una forza sufficiente per muovere il robot anche a velocità elevate (fino a circa 1.5 m/sec);
- Due cuscinetti bloccanti Koyo model UP000;
- Una ruota omni-direzionale Opteq del diametro di 100mm;

- Un giunto elastico Item, utilizzato per collegare l'albero di trasmissione alla ruota, e per ridurre la trasmissione di sollecitazioni meccaniche.

La struttura cosi composta permette al robot di compiere movimenti fluidi e di muoversi in piena sicurezza nell'ambiente. Questa struttura ha sostituito la precedente base aumentandone la stabilità, e migliorando le prestazioni complessive di navigazione.

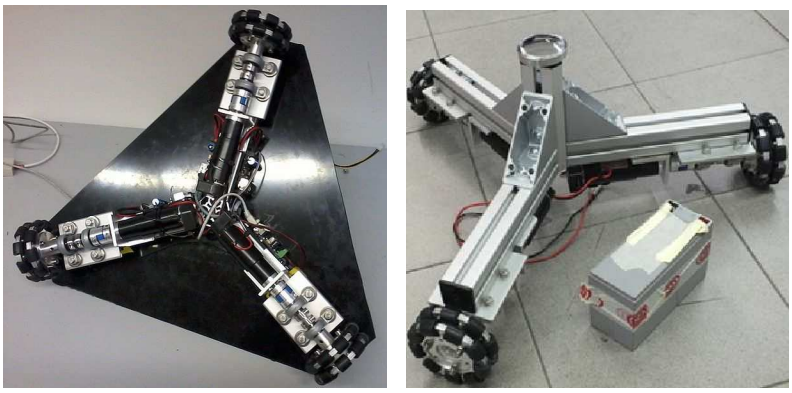


Figura 3.2: Base modulo Triskar

3.1.2 Elettronica

Gli elementi elettronici della base sono basati sull'architettura R2P [9], un approccio modulare sviluppato da Martino Migliavacca all'interno di AIRLab. Le schede sono indipendenti le une dalle altre e sono state progettate tenendo in considerazione il costo di realizzazione, ognuna di esse implementa una singola funzionalità della base (controllo dei motori, comunicazione, interfaccia sensori, interfaccia Ethernet, ...); esse sono collegate attraverso CAN bus, in modo da partecipare ad una architettura publisher-suscriber supportata dal sistema operativo real-time ChiBiOS [14]. Questo permette che l'interazione tra le varie schede sia espletata in real-time. Ognuna di esse include un processore ARM, intorno al quale sono sviluppate le singole funzionalità del modulo.

3.1.3 Calcolatore

Il robot è dotato di un mini-pc Shuttle DS81 supportato da un processore Intel Core i7-4770S (3.10GHz), 8GiB di memoria ram e un Harddisk da 60GiB su cui è installato il sistema operativo (Linux Ubuntu 14.04 LTS). Tale piattaforma riesce a garantire una sufficiente potenza computazionale senza sacrificare eccessivamente spazio e consumi.

3.1.4 Collo e Testa

Il collo è progettato come una struttura mobile con differenti gradi di libertà ed è costituita da cinque moduli (Fig. 3.4) che ne permettono il movimento, conferendo allo stesso tempo solidità e sicurezza all'intero sistema. La Figura 3.3 documenta una rappresentazione grafica della struttura del collo. Il modulo superiore e quello inferiore sono identici ed insieme forniscono una inclinazione di 30° nei movimenti posteriori e di 60° per gli spostamenti in avanti. I moduli adiacenti, vengono utilizzati per permettere movimenti di lateralità con una inclinazione di 30°, sia per il lato destro che per quello sinistro. Da ultimo il modulo intermedio è utilizzato per accompagnare gli spostamenti frontali e posteriori, conferendo un movimento più naturale.

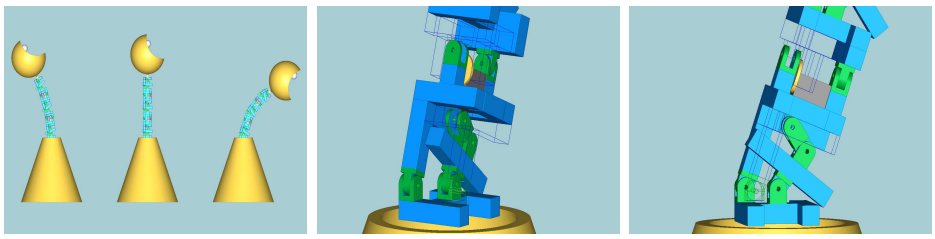


Figura 3.3: Visualizzazione grafica collo E-2?

La testa del robot contiene 4 servomotori per muovere gli occhi, 2 per le sopracciglia, uno per la bocca ed uno per favorire la rotazione della testa di E-2?. Al suo interno è posizionata la scheda utilizzata per il controllo dei servomotori, la Pololu Mini Maestro. Il dispositivo permette di memorizzare differenti posizioni nel tempo degli attuatori in grado di replicare azioni che si traducono in espressioni facciali e movimenti del collo.

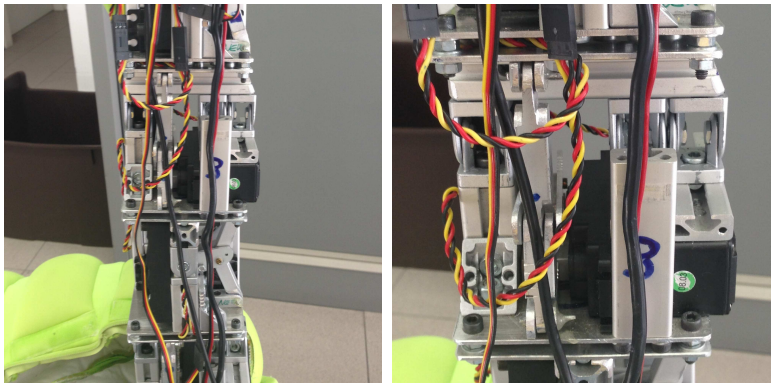


Figura 3.4: Moduli collo E-2?

3.1.5 Dispositivi Audio

Il robot si avvale di un unico altoparlante portatile Western WS-Q9. Quest'ultimo sostituisce gli speaker precedentemente montati sulla macchina migliorando l'ingombro e la resa dell'audio e favorendo un risparmio a livello energetico, garantito dalla batteria interna al dispositivo.

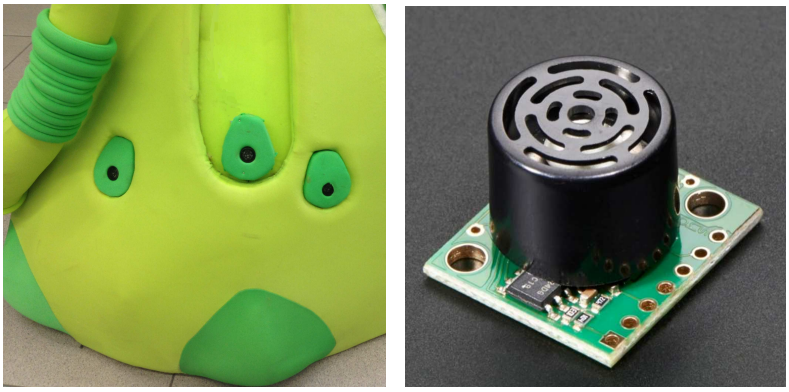
Figura 3.5: Altoparlante WSTER WS-Q9

3.1.6 Alimentazione

Il robot utilizza una corrente continua a 24V, basata su una coppia di batterie al piombo da 12V collegate in serie, per alimentare i motori della base; un'altra coppia alimenta il computer di bordo ed i servomotori del collo di E-2?. Le ulteriori due batterie da 12V alimentano rispettivamente il sensore Kinect e il router Linksys utilizzato per il collegamento remoto con il robot. La ripartizione cosi effettuata garantisce un'autonomia di diverse ore a pieno utilizzo.

3.1.7 Sonar

E-2? è equipaggiato anche di una cintura sonar utilizzata per misurare la distanza tra un oggetto ed il trasmettitore stesso. La cintura, sviluppata ed assemblata in AIRLab, è dotata di 7 sensori Maxbotix EZ-2 (Fig. 3.6b), collegati ad un micro-controllore che si occupa di sequenziare i sensori e trasmettere ai nodi le distanze misurate.



(a) Distribuzione sensori *(b) Sensore Maxbotix EZ-2*

Figura 3.6: Sonar E-2?

I dispositivi utilizzati, sono in grado di rilevare la presenza di oggetti o persone ad una distanza compresa tra 0 e 6 metri. Questi sensori vengono utilizzati per controllare se una persona è situata accanto al robot, mentre questo sta procedendo in direzione dello stand. Le limitazioni fisiche dei dispositivi, quali il raggio di trasmissione e ricezione degli impulsi (ca. 20°), hanno richiesto un riposizionamento dei sensori che sono stati situati a coppie di tre ai lati del corpo (Fig. 3.6a), lasciando un solo sensore anteriormente; quest'ultimo viene utilizzato come sensore aggiuntivo al rilevamento di ostacoli frontali.

3.1.8 Kinect

Il robot infine è equipaggiato con un sensore Kinect posizionato sopra la testa del robot. Originariamente conosciuto con il nome in codice di Progetto Natal, 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.

Figura 3.7: Microsoft Kinect

La scelta di utilizzare Kinect è da ricercarsi nelle potenzialità di motion sensing di tale dispositivo. Questo apparato mette a disposizione numerosi sensori spesso utilizzati nell'ambito della visione artificiale quali una camera RGB, un sensore di profondità, una camera ad infrarossi, 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 migliore dei modi il volto dell'utente con cui il robot dovrà 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 è di circa 57° sul piano orizzontale e 43° sul piano verticale; il piede motorizzato, infine, permette di inclinare il dispositivo di un angolo di 60° rispetto all'asse verticale. Inoltre il sensore viene utilizzato per ricevere informazioni riguardo all'ambiente e agli ostacoli, permettendo al robot di localizzarsi, senza dover utilizzare costosi sensori quali laser-scanner.

Il corpo del sensore, contenuto in una barra orizzontale, è connesso ad un piccolo piedistallo motorizzato ideato per essere posizionato sopra o sotto il televisore utilizzato con la console. Esso contiene una camera RGB, un sensore di profondità, un accelerometro ed un array di microfoni. Il controllo avviene tramite firmware proprietario che consente il riconoscimento e l'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 ad un sensore CMOS monocromatico, il quale permette di catturare dati video in 3D sotto qualsiasi condizione di luce.

L'array di microfoni formato da 4 periferiche, processa un segnale audio a 16-bit @16KHz per ogni canale. Il dispositivo è infine alimentato tramite connessione USB supportata da un'alimentazione di 12V aggiuntiva, che ha richiesto l'inserimento di un'ulteriore batteria all'interno del robot. La principale innovazione proposta dal Kinect 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 fino ad 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.

Questo sensore si presta molto bene quindi per le elaborazioni richieste dal progetto ed è pienamente supportato dall'ambiente ROS [4] su cui è stato sviluppato il software di controllo.

3.2 Software

In questa sezione verrà descritto ROS (Robot Operating System), ambiente utilizzato per lo sviluppo del sistema di controllo del robot, intorno al quale sono state sviluppate le varie componenti dell'architettura software. Successivamente verrà presentata l'architettura software del sistema di controllo.

3.2.1 ROS: Robot Operating System

ROS 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, scambio di messaggi tra processi ed un sistema di gestione dei propri pacchetti. A questo si aggiungono tool e librerie

per la scrittura, compilazione ed esecuzione di codice anche in forma distribuita su più computer.

Da un punto di vista concettuale ROS può essere diviso in tre livelli:

- **Filesystem level**: definisce uno standard per la gestione e la creazione dei moduli che comporranno il sistema di controllo;
- **Computation Graph level**: il grafo computazionale è una rete peer-to-peer di processi ROS che si scambiano messaggi tra loro, e può considerarsi come il cuore dell'architettura ROS;
- **Community level**: una serie di strumenti messi a disposizione per favorire lo scambio di conoscenze e software, e per il mantenimento dei pacchetti all'interno della community;

Il Computation Graph è una rete peer-to-peer dei 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 per questo motivo ogni singolo nodo dovrebbe occuparsi di un ristretto numero di obiettivi, come il controllo delle ruote, o i dati provenienti da un sensore come un laser o un sonar. Per la scrittura dei nodi vengono messe a disposizione due librerie: *roscpp* e *rospy* rispettivamente per il linguaggio C++ e Python;
- **Master**: il nodo Master fornisce metodi per la ricerca e la registrazione a tutti i nodi che fanno parte del *Computational Graph*. Senza questo i nodi non sarebbero capaci di trovarsi, scambiarsi messaggi o invocare servizi;
- **Messaggi**: i nodi comunicano tra di loro mediante lo scambio di messaggi. Questi non sono altro che semplici strutture dati composte da campi tipizzati. Vengono supportati tutti i principali tipi primitivi (integer, float, boolean, etc), array degli stessi, e strutture nidificate come accade nel linguaggio C;
- **Topics**: i messaggi precedentemente descritti vengono indirizzati attraverso un sistema di trasporto con una semantica publish-suscribe. Un nodo può mandare un messaggio pubblicandolo su

un determinato topic. Il topic quindi non è altro che un nome che identifica il contenuto del messaggio nella rete ROS. Ne segue che un nodo, interessato ad un determinato tipo di dato, non deve far altro che sottoscriversi al topic di interesse per ricevere i messaggi di cui ha bisogno. Possono esserci più nodi che pubblicano messaggi sullo stesso topic come anche molti suscriber. Il topic, in definitiva, può essere pensato come un message-bus specifico di una fonte di dati, in cui ogni nodo può mandare o ricevere messaggi su quel determinato topic;

- **Servizi**: l'architettura publisher-suscriber è un paradigma di comunicazione molto flessibile, ma non è adatto ad interazioni del tipo richiesta-risposta, che sono spesso necessarie in un sistema distribuito. Queste interazioni vengono gestite con l'ausilio dei servizi che vengono esportati dai nodi. Ogni nodo può definire un determinato numero di servizi a seconda delle esigenze con un dato nome. Il cliente che richiede il servizio resta poi in attesa fino al completamento dell'azione ;

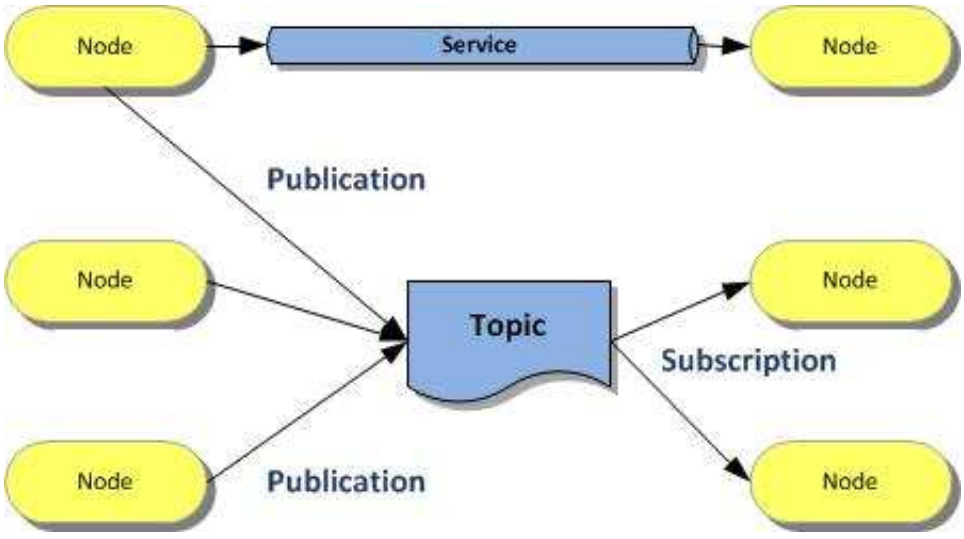


Figura 3.8: Esempio di un sistema ROS

- **Bags**: il bag è un formato definito all'interno di ROS per salvare i dati dei messaggi. Sono un utile meccanismo per memorizzare informazioni, effettuare prove su algoritmi e per testare il funziona-

mento dei pacchetti creati, senza dover per forza agire fisicamente sul robot.

L'architettura così presentata permette quindi di sviluppare un sistema di controllo anche molto complesso e facilmente estendibile, attraverso la creazione di semplici nodi che utilizzino la struttura di comunicazione appena descritta, facilitando la fase di sviluppo e di testing.

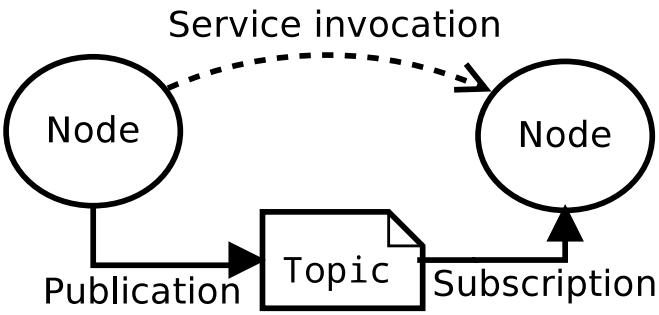


Figura 3.9: Architettura Publisher-Subscriber

3.2.2 Architettura software

Come descritto nel paragrafo precedente, l'intero sistema di controllo adotta lo stile tipico del framework ROS, in modo che ogni pacchetto gestisca una singola funzionalità del robot. Concettualmente l'architettura può essere suddivisa in quattro strati:

1. Controllo
2. Navigazione
3. Analisi Facciale
4. Astrazione Hardware

I nodi che si occupano di gestire le componenti hardware del robot sono stati sviluppati utilizzando l'interfaccia *actionlib*. I motivi di questa scelta sono da ricercarsi nella necessità di ricevere costantemente informazioni riguardo lo stato interno dei nodi, sicché il robot sia sempre al corrente dello stato delle attività. In particolare, questa libreria permette di creare un paradigma client-server come mostrato in Figura 3.10, attivando una comunicazione attraverso il *ROS Action*

Protocol, definito sopra lo stack dei messaggi ROS [1]. Il paradigma permette di avere un nodo dedicato che si comporta come un server, che resta in attesa di ricevere un goal proveniente da uno o più client che ne fanno richiesta. In tal modo più nodi possono effettuare richieste simultaneamente allo stesso nodo, senza preoccuparsi che le loro richieste si sovrappongano o vengano perse.

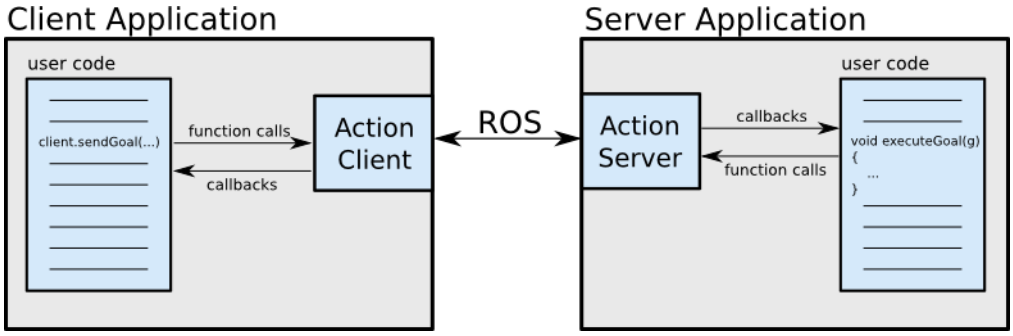


Figura 3.10: Architettura interfaccia Actionlib

Il sistema di controllo consta di un singolo nodo che ha lo scopo di pianificare le azioni che il robot dovrà compiere per perseguire i suoi obiettivi. Il nodo in questione è *e2_brain*, il quale decide valutando informazioni quali distanza, interesse della persona e stato interno dei nodi, quale sia la migliore azione da compiere. Infine lo stack si compone di una serie di pacchetti di configurazione e di avvio, necessari per il corretto funzionamento dell'intero sistema.

A seguire, lo stack di navigazione si compone di tutti quei nodi che supportano il robot nella movimentazione nello spazio di lavoro. I pacchetti che ne fanno parte sono:

- **e2_navigation**: questo nodo riceve i task di navigazione provenienti dal modulo *e2_brain*, pianifica e controlla le fasi di ricerca ed approccio verso una persona e la successiva fase di navigazione verso lo stand obiettivo.
- **e2_nav_messages**: il robot, una volta pianificata una traiettoria, trasmette una serie di messaggi sotto forma di velocità lineari e angolari per seguire correttamente la via stabilita. Il compito di questo nodo è di convertire questi messaggi in un formato che

sia comprensibile dalla base omnidirezionale, permettendone il movimento.

- **e2_odometry**: il nodo riceve costantemente informazioni provenienti dagli encoder delle ruote, aggiornando periodicamente la posizione del robot nello spazio. Può essere utilizzato in due modi differenti, calcolando la posizione mediante i dati degli encoder, o utilizzando i messaggi di velocità generati dal pianificatore nel caso i dati dagli encoder non siano disponibili, ad esempio a seguito di un guasto. Quest'ultimo nodo ha un importanza cruciale per il corretto funzionamento del sistema di navigazione: senza un'accurata informazione sul suo posizionamento, il robot si sposterebbe seguendo una traiettoria non corretta.

La sezione di astrazione dell'hardware raccoglie i nodi che controllano i vari sistemi hardware del robot. Ognuno di questi implementa le actionlib, in modo da poter essere richiamati facilmente dai sistemi che ne fanno richiesta. I pacchetti di cui si compone sono:

- **e2_neck_controller**: tale modulo esporta tutte le funzionalità del collo e del volto di E-2, permettendo di richiamare movimenti ed espressioni pre-programmate sulla Mini Maestro Pololu.

- **e2_voice**: questo nodo riceve messaggi di testo che vengono processati e trasformati in file audio per essere poi riprodotti dal robot, consentendo il dialogo con la persona. Il modulo implementa la libreria SVOX garantendo una migliore resa audio e una maggiore fluidità del linguaggio. A causa della mancanza di una voce maschile, è stato inoltre applicato un filtro audio real-time per rendere la voce femminile disponibile, in una più mascolina e profonda.

- **e2_sonar**: interpreta i dati ricevuti dai sensori ad ultrasuoni posti attorno alla base del robot, per poi ritrasmetterli ai nodi che ne fanno richiesta, tramite pubblicazione all'interno del topic *e2_sonar*. Queste informazioni sono un utile strumento per determinare la posizione della persona durante la fase di avvicinamento verso lo stand obiettivo.

- **kinect_motor**: si occupa di controllare il motore del sensore Kinect, modificandone l'inclinazione verticale a seconda delle esigenze.

Lo stack di analisi facciale permette di condurre due tipi di analisi dei volti delle persone con cui il robot interagisce. Le analisi disponibili sono di due tipi: la prima permette di determinare l'interesse di una persona a seguito di una serie di affermazioni di E-2?, basato sullo studio dei movimenti muscolari della faccia, mentre la seconda aiuta a rilevare la stessa persona nel corso del tempo, dotando il robot di una memoria visiva. I pacchetti che ne fanno parte sono descritti qui di seguito:

- **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.

- **face-recognition**: modulo ROS che permette al robot di memorizzare il volto di una persona e di riconoscerlo nell'arco dell'intero processo di interazione.

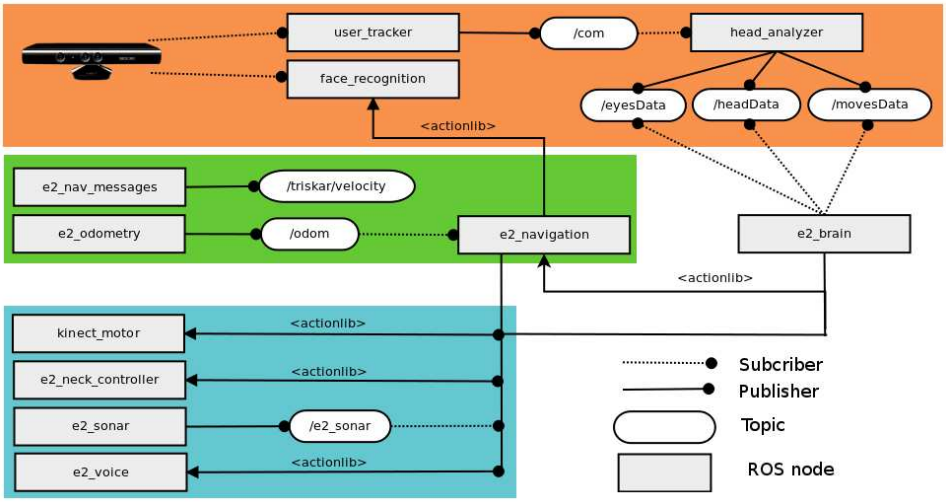


Figura 3.11: Architettura del sistema di controllo

Infine è stato sviluppato un modello 3D del robot e un ambiente simile a quello di una fiera, che permette di simulare il funzionamento dei vari sotto-sistemi all'interno simulatore V-Rep. All'interno del pacchetto e2_simulator si trovano i nodi che consentono di interfacciarsi con l'architettura ROS.

- **base-controller**: trasforma i messaggi provenienti dallo stack di navigazione in movimenti del robot simulato, controllandone la velocità delle ruote.
- **fake-odom**: vengono generati i dati riguardo la posizione del robot, utilizzando le informazioni provenienti dal simulatore.
- **depthbuffer-to-img**: questo nodo è stato creato per risolvere una limitazione del simulatore, che non era in grado di riprodurre lo stream di profondità del Kinect.

Per una maggiore documentazione a riguardo dell'ambiente di simulazione e dell'utilizzo dei pacchetti si rimanda all'appendice A che contiene la descrizione dettagliata del software prodotto e le istruzioni per l'utilizzo del simulatore.

Capitolo 4

Sistema di navigazione

"Da soli possiamo fare così poco; insieme possiamo fare così tanto."

Helen Adams Keller

Una delle novità del lavoro svolto nel corso di questo progetto di tesi, è caratterizzata dall'aggiunta al sistema precedente dello stack di navigazione, che permette al robot di localizzarsi e muoversi liberamente nell'ambiente in cui opera. La realizzazione di questa funzionalità ha evidenziato una serie di problematiche, che sono state affrontate per poter ottenere dei buoni risultati. Un aiuto alla risoluzione di alcune problematiche, è stato dato dalla piattaforma ROS, che ha semplificato alcuni compiti complessi, mettendo a disposizione una serie di pacchetti altamente configurabili, che consentono di gestire fasi cruciali della navigazione, senza sacrificare troppo costi e performance in fase di utilizzo.

Affinché il robot possa spostarsi in piena autonomia, questi deve comprendere l'ambiente che lo circonda attraverso la rappresentazione dei dati catturati dai sensori, deve inoltre essere consapevole della sua posizione sulla mappa per poter pianificare una traiettoria su cui muoversi. L'ambiente in cui opera il robot E-2?, quello di uno spazio espositivo, è altamente dinamico, poiché possono comparire in breve tempo, oggetti o persone nel campo di percezione, ostacolandone la traiettoria di navigazione. Quest'ultimo problema implica la necessità di poter prevedere ostacoli ed evitarli, aggiungendo un ulteriore livello di complessità al sistema stesso.

Di seguito viene prima presentato lo stack di navigazione implementato in ROS, vengono descritte le operazioni effettuate durante un task di navigazione e successivamente vengono mostrate le componenti che si inseriscono accanto a questa architettura per completare la fase di controllo.

4.1 Navigazione

Lo stack di navigazione si compone di più nodi, i quali collaborano tra di loro, scambiandosi informazioni sullo stato del robot, la sua posizione e la percezione dell'ambiente. In Figura 4.1 viene mostrata l'architettura ROS che ha il compito di pianificare il movimento della base, evidenziando i nodi coinvolti nell'operazione: il modulo *amcl* fornisce i dati riguardo il posizionamento del robot; il *map_server* si occupa di tener traccia della mappa in cui il robot opera, aggiornando le informazioni riguardanti eventuali ostacoli; infine il nodo *move_base* ha il compito di pianificare la traiettoria di movimento utilizzando i dati provenienti dal sensore Kinect e dal nodo *amcl*.

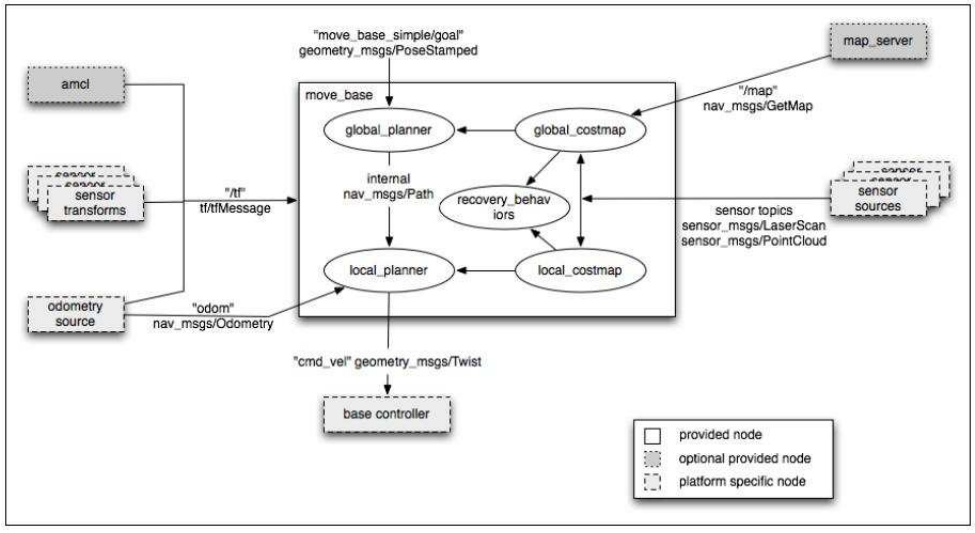


Figura 4.1: Navigation overview

Lo stack di navigazione tiene traccia della posizione del robot utilizzando una rappresentazione di frame di coordinate che determinano

l'orientamento, e la posizione di un oggetto, rispetto ad un frame statico. In questo modo è possibile avere costantemente una relazione tra il robot e la mappa dell'ambiente [16]. Il robot utilizza una mappa statica, generata mediante telecontrollo, la quale viene aggiornata a seguito della ricezione dei dati provenienti dal Kinect, utilizzando il sensore di profondità che trasforma le distanze percepite in una mappa 2D mediante algoritmi di *SLAM* (Simultaneous localization and mapping).

4.2 Localizzazione

Definito l'ambiente di lavoro è necessario stabilire dove si trova il robot sulla mappa, determinandone la posizione e orientamento della base. Il problema della localizzazione è uno dei più importanti sui robot mobili, a causa di fattori che possono presentarsi sotto diverse forme. Ad esempio un attuatore non perfettamente collegato ad una ruota genera un errore dovuto allo slittamento dell'asse prima che intervenga la frizione dei due oggetti per far muovere la ruota. Un altro errore quasi sempre presente è causato dalle approssimazioni nei calcoli causati dalle operazioni di differenziazione e integrazione, ed ancora l'utilizzo di dati quantizzati, e quindi non precisi, che con il passare del tempo incrementa l'errore di posizionamento. Esistono tuttavia diverse soluzioni per risolvere questi problemi, ognuna con pregi e difetti, ed il sistema utilizzato varia di norma in base ai sensori disponibili sul robot e in base alle necessità di precisione richieste.

Per lo scopo dell'elaborato era richiesto un buon grado di precisione dato che il robot deve operare in piena autonomia nell'ambiente, evitando ostacoli e persone. La tecnica implementata prevede un sistema di controllo misto che si compone di due componenti, una di Dead Reckoning, che utilizza i dati odometrici per calcolare lo spostamento del robot, ed un sistema di localizzazione probabilistico **AMCL**, che aiuti a migliorare la corretta predizione della posizione. La scelta di utilizzare questo sistema garantisce un sistema robusto e facilmente adattabile ad altri robot, senza richiedere elevate risorse computazionali o sensori dal costo elevato; quest'ultima richiesta era una prerogativa del progetto. Per questo è stato utilizzato un semplice sensore Kinect che si affiancasse agli encoder del robot, evitando l'utilizzo di sensori più pre-

cisi quali laser scanner, di costo molto superiore. Di seguito vengono descritti i due sistemi che collaborano per effettuare la localizzazione del robot.

4.2.1 Odometria

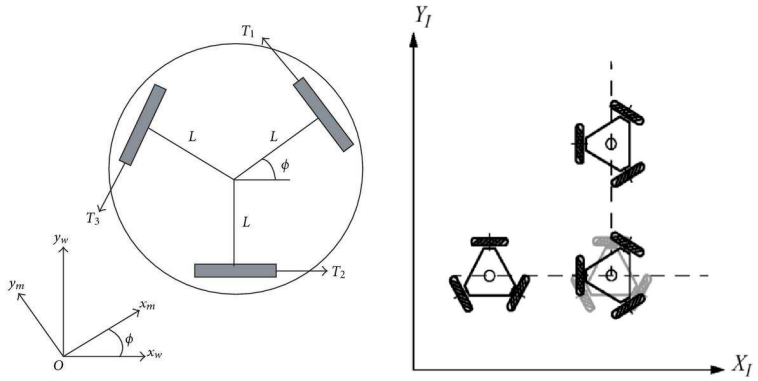
L'odometria è una tecnica usata per stimare la posizione di un veicolo su ruote che si basa su informazioni provenienti da sensori interni al robot per calcolare lo spazio percorso e l'orientamento. I sensori utilizzati nel progetto sono tre encoder incrementali, montati a monte dell'asse motore e che forniscono le velocità in tempo reale delle ruote di E-2?. In questo modo il robot è in grado di valutare la distanza percorsa, risultando una componente importante anche nell'approccio ad ostacoli presenti nell'ambiente [20].

Tuttavia a causa della natura quantizzata dei valori ricevuti nel tempo, le operazioni di differenziazione e integrazione non possono essere applicate senza incorrere in approssimazioni che si traducono in un errore di posizionamento. Per effetto degli errori che si accumulano nella navigazione la postura (x, y, θ) del robot nell'ambiente diventa rapidamente diversa da quella stimata. Questo problema è stato risolto, parzialmente, utilizzando un metodo di calcolo diretto che eliminasse le approssimazioni dovute a integrazioni e differenziazioni nei calcoli, ottenendo un risultato più preciso. In Figura 4.2a si vede una rappresentazione della base sul piano cartesiano. Definiamo con **L** la distanza che intercorre tra la ruota e il centro di massa del robot; con **r** il raggio di una ruota del robot, e con φ_1, φ_2, φ_3 le posizioni lette dai tre encoder.

La formulazione utilizzata [25] [26] permette di calcolare direttamente i valori di x, y e θ utilizzando come parametro il vettore delle velocità delle ruote. Il calcolo fa uso delle equazioni di moto e richiede il calcolo della formula:

$$\dot{S} = r \int W^{-1}.\dot{\Phi} dt \tag{4.1}$$

Dove



(a) Struttura Base *(b) Movimenti Base*

Figura 4.2: E-2? Rappresentazione base mobile

$$S = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} x\left(\varphi_1, \varphi_2, \varphi_3\right) \\ y\left(\varphi_1, \varphi_2, \varphi_3\right) \\ \theta\left(\varphi_1, \varphi_2, \varphi_3\right) \end{bmatrix} \tag{4.2}$$

$$W = \frac{1}{r} \begin{bmatrix} -\sin(\theta) & \cos(\theta) & L \\ -\sin(\frac{\pi}{3} - \theta) & -\cos(\frac{\pi}{3} - \theta) & L \\ \sin(\frac{\pi}{3} + \theta) & -\cos(\frac{\pi}{3} + \theta) & L \end{bmatrix} \tag{4.3}$$

Sviluppando i calcoli si ottiene un sistema di equazioni così definito

$$\begin{aligned} x &= r \int \frac{1}{3} \sin(\frac{\pi}{3})[(\cos(\frac{\pi}{3} + \theta) - \cos(\frac{\pi}{3} - \theta))\dot{\varphi_1} + \\ &\quad (-\cos(\theta o - \cos(\frac{\pi}{3} + \theta))\dot{\varphi_2} + (\cos(\theta c - \cos(\frac{\pi}{3} - \theta))\dot{\varphi_3}]\, dt \\ y &= r \int \frac{1}{3} \sin(\frac{\pi}{3})[(\sin(\frac{\pi}{3} - \theta) - \sin(\frac{\pi}{3} + \theta))\dot{\varphi_1} + \\ &\quad (-\sin(\theta i - \sin(\frac{\pi}{3} + \theta))\dot{\varphi_2} + (\sin(\theta s - \sin(\frac{\pi}{3} - \theta))\dot{\varphi_3}]\, dt \\ \theta &= r \int \frac{\dot{\varphi_1} + \dot{\varphi_2} + \dot{\varphi_3}}{3L}\, dt \end{aligned} \tag{4.4}$$

Ipotizzando che θ sia indipendente nel tempo, le formule sopra descritte possono essere semplificate e sviluppate come:

$$
\begin{aligned}
x = \frac{r}{3\sin\frac{\pi}{3}}[&(\cos(\frac{\pi}{3}+\theta)-\cos(\frac{\pi}{3}-\theta)]\dot{\varphi}_1+ \\
&(-\cos\theta-\cos(\frac{\pi}{3}+\theta))\dot{\varphi}_2+(\cos\theta-\cos\frac{\pi}{3}-\theta)\dot{\varphi}_3]\,dt \\
y = \frac{r}{3\sin\frac{\pi}{3}}[&(\sin(\frac{\pi}{3}-\theta)+\sin(\frac{\pi}{3}+\theta))\dot{\varphi}_1+ \\
&(-\sin\theta-\sin(\frac{\pi}{3}+\theta))\dot{\varphi}_2+(\sin(\theta-\sin(\frac{\pi}{3}-\theta))\dot{\varphi}_3]\,dt
\end{aligned}
\tag{4.5}
$$

$$
\theta = \frac{1}{3\,L}(\dot{\varphi}_1+\dot{\varphi}_2+\dot{\varphi}_3) \tag{4.6}
$$

Le equazioni mostrano che la posizione del robot può essere estrapolata direttamente se il robot non compie rotazioni mentre si muove verso la nuova posizione. Se θ cambia il suo valore da zero, l'origine del sistema di coordinate deve essere spostato rispetto alla nuova variazione e si può procedere al calcolo. Nonostante le equazioni facciano l'assunzione che θ sia costante, θ viene sostituito nelle formule 4.5 con l'angolo calcolato dalla 4.6. Con il metodo appena descritto, viene notevolmente ridotto il presentarsi di errori di accumulazione rispetto ad un approccio differenziale.

4.2.2 Il modulo e2_odometry

Il codice necessario al calcolo è implementato nel pacchetto *e2_odometry* che recupera i dati dagli encoder ed utilizza questi dati per determinare la nuova posizione del robot. Nel listato 4.1 si può vedere come i dati ricevuti dagli encoder vengano sincronizzati prima di poter effettuare il calcolo, mediante la libreria *message_filter*.

```
message_filters::Subscriber<EncoderStamped> sub_enc_1(nh, enc_1, 1);
message_filters::Subscriber<EncoderStamped> sub_enc_2(nh, enc_2, 1);
message_filters::Subscriber<EncoderStamped> sub_enc_3(nh, enc_3, 1);

TimeSynchronizer<EncoderStamped,EncoderStamped,EncoderStamped> sync(
    sub_enc_1,sub_enc_2,sub_enc_3, 10);

sync.registerCallback(boost::bind(&encoderCallback, _1, _2,_3));

ros::Subscriber sub_cmd_vel= nh.subscribe(vel_topic, 10, getRobotVelocity);
```

```
ros::Publisher odom_pub = nh.advertise<nav_msgs::Odometry>("/odom", 1);
```

Listing 4.1: Sincronizzazione messaggi encoder

Una volta ricevuti tutti i dati necessari, viene espletato il calcolo della nuova posizione mediante le formule 4.5 e 4.6. I valori ottenuti vengono quindi trasmessi sul topic */odom* e verranno in seguito utilizzati dal modulo AMCL per stimare la posizione del robot sulla mappa, associando queste informazioni con quelle provenienti dal sensore Kinect.

4.2.3 AMCL

AMCL (Adaptive Monte Carlo Localization) [18] è un sistema di localizzazione probabilistico per robot che si muovono su mappe 2D. Questo pacchetto implementa l'approccio Monte Carlo, che utilizza un filtro di particelle per determinare la posa del robot rispetto ad una mappa, utilizzando i dati provenienti dai sensori, in questo caso Kinect, per effettuare una stima riguardo alla posizione del robot. L'utilizzo di questo sistema deriva dal fatto che il robot non può sapere con certezza la sua posizione attuale, per questo mediante un algoritmo vengono generate delle ipotesi sulla posizione in cui si troverà il robot in seguito ad uno spostamento. Queste supposizioni vengono chiamate particelle, ed ognuna di queste rappresenta una possibile posa del robot indicando posizione e orientamento tramite un vettore sulla mappa (Fig. 4.3). Ogni particella contiene quindi una descrizione completa di un possibile stato futuro, e ogni qual volta il robot osserva l'ambiente, attraverso i dati di Kinect, scarta le particelle inconsistenti con quell'osservazione e genera un numero di particelle che appaiono coerenti con i dati osservati. Ne risulta che la maggior parte delle particelle converge alla posizione in cui il robot si trova realmente.

Il nodo è integrato in ROS attraverso un sistema di messaggi che contengono la trasformata dal frame map frame al frame odom (Figura 4.4), che in conclusione migliora la stima della posa del robot con l'ausilio dei dati odometrici, migliorandone la localizzazione.

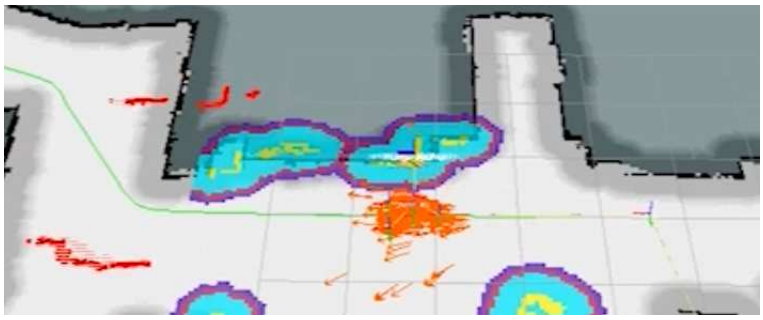


Figura 4.3: Le frecce arancioni rappresentano le ipotesi del robot

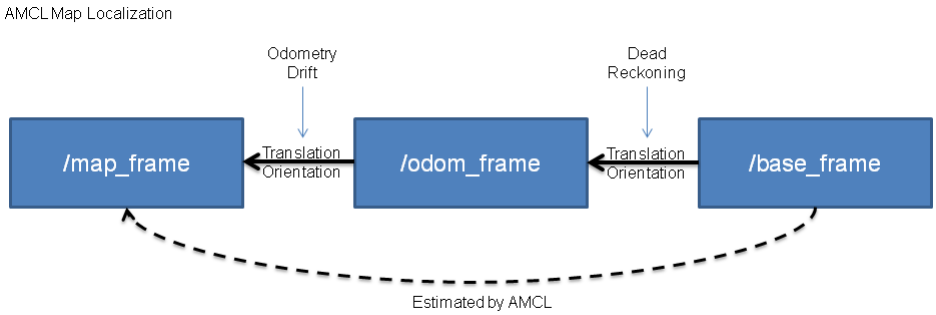


Figura 4.4: Architettura modulo AMCL

4.3 Path Planning

Con il termine Path Planning, si identificano tutte quelle tecniche che hanno lo scopo di determinare un percorso che vada da un punto A ad un punto B.

Il robot si avvale di tre moduli per programmare le traiettorie da seguire: un *global planner*, un *local planner* e un *recovery behaviours* (Figura 4.5). Il primo viene utilizzato per generare il percorso che dovrà seguire il robot a seguito della ricezione di un punto obiettivo, mentre il secondo verrà utilizzato per generare la cinematica di movimento che permetta alla base di seguire la traiettoria precedentemente generata. Infine il modulo *recovery behaviors* gestisce le situazioni di stallo, in cui il robot è impossibilitato a muoversi, generando delle traiettorie per liberarsi e riprendere il percorso originario.

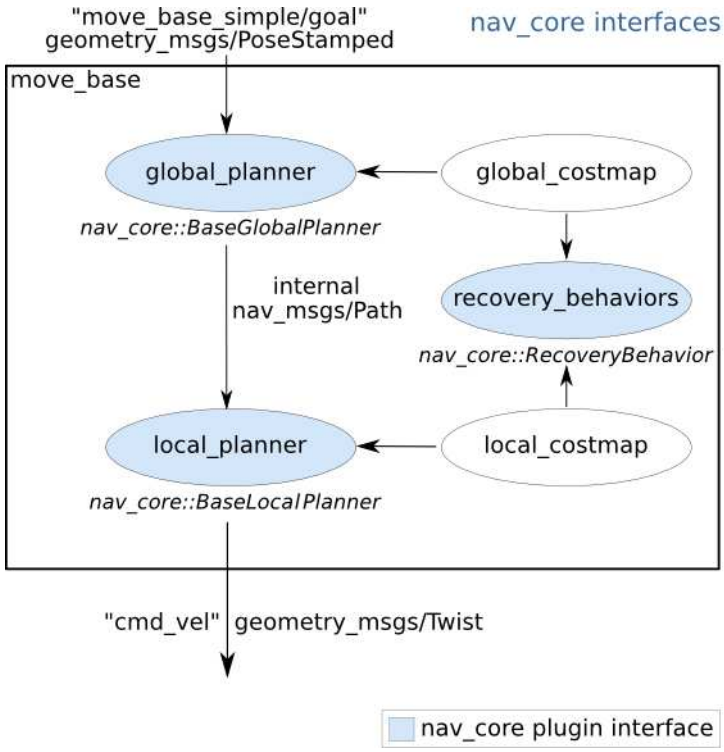


Figura 4.5: Architettura Path Planner

4.3.1 Global Planner

Il Global Planner fornisce un interfaccia per pianificare una traiettoria globale che il robot dovrà seguire per raggiungere un punto obiettivo. Tuttavia, il percorso generato a volte risulta troppo ottimistico; ad esempio una traiettoria potrebbe trovarsi in un uno spazio non sufficientemente ampio da permettere il passaggio del robot. Come planner globale è stato utilizzato *navfn* [2]. Il pianificatore opera su una costmap per trovare un piano di costo minimo da un punto iniziale a un punto finale su una griglia in cui sono riportati, sotto forma di costi, ostacoli e muri, ed utilizzando l'algoritmo di Dijkstra viene pianificato un percorso di costo minimo da seguire.

4.3.2 Local Planner

Il local planner provvede al controllo e alla movimentazione della base mobile nello spazio. Questo pianificatore può essere visto come uno strato che collega il global planner al robot. Il pianificatore crea una

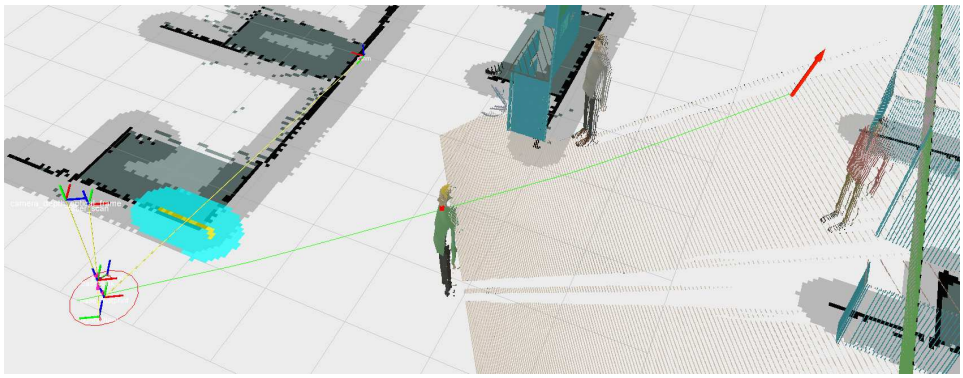


Figura 4.6: In verde la traiettoria generata dal global planner

traiettoria cinematica affinché il robot possa muoversi, sotto forma di messaggi di velocità interpretabili dalla base. Questo modulo agisce su una mappa di dimensioni ridotta con il robot al centro di essa, ed utilizza una funzione di valore basata sul costo per determinare le velocità di traslazione e di rotazione da seguire.

L'algoritmo utilizzato è il DWA (Dynamic Window Approach) [17], del quale viene riportato schematicamente il funzionamento:

1. Viene fatto un campionamento di velocità nello spazio di controllo del robot (dx,dy,dtheta);
2. Per ogni velocità campionata, viene effettuata una simulazione dallo stato corrente del robot per prevedere cosa accadrà se la velocità in esame fosse applicata per un breve periodo di tempo;
3. Viene valutata ogni traiettoria ottenuta dalla simulazione, utilizzando una metrica che incorpora caratteristiche come: prossimità degli ostacoli, vicinanza all'obiettivo, vicinanza alla traiettoria globale e velocità;
4. Vengono eliminate le traiettorie che generano collisioni;
5. Viene scelta la traiettoria con la valutazione più alta e vengono inviate le relative velocità, attraverso messaggi alla base mobile;
6. Si ricomincia il processo dal punto 1 finché non viene raggiunto l'obiettivo.

4.4 Recovery Behaviours

Il robot è configurato per utilizzare dei comportamenti da usare in caso di situazioni di stallo o se non è nelle condizioni di seguire la traiettoria prestabilita. Vengono eseguite le seguenti azioni per tentare di liberare il robot da queste situazioni. In prima istanza, gli ostacoli esterni alla mappa locale vengono eliminati per poter essere aggiornati da nuove informazioni provenienti dal sensore Kinect. In seguito, se possibile, il robot tenterà di effettuare una rotazione sul posto per controllare se sono ancora presenti gli ostacoli che ne impediscono il movimento. Se tutte queste azioni falliscono, il robot considera il suo goal non raggiungibile e notifica il fallimento della navigazione ai sistemi di controllo.

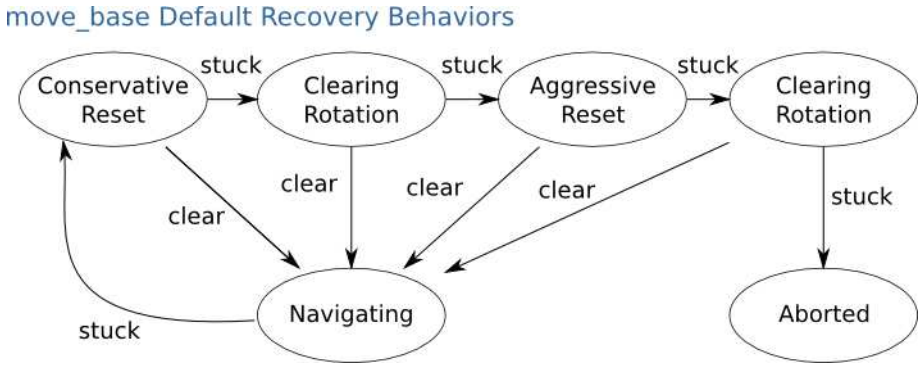


Figura 4.7: Recovery Behaviours

4.5 Il modulo e2_navigation

Il modulo *e2_navigation* è stato sviluppato per supportare il robot durante il processo di navigazione, gestendo le fasi di tracking e avvicinamento alle persone, e pianificando le azioni da compiere mentre procede verso lo stand obiettivo. Il modulo può essere attivato utilizzando i servizi abilitati tramite interfaccia ROS o attraverso i messaggi ricevuti dal modulo *e2_brain* che verrà descritto nel Capitolo 6.

La definizione delle azioni disponibili di tale modulo è riportata all'interno del file *Nav.action*, contenuto all'interno della cartella *action* del modulo in analisi.

```
#goal definition
uint8  action_id
---
#result definition
uint8  action_id
string result
---
#feedback
uint8  action_id
string status
```

Listing 4.2: Implementazione Nav.action

Come si può osservare dal dettaglio implementativo riportato sopra, il goal di tale azione è definito ricorrendo alla variabile *action_id*, che determina quale azione deve compiere il modulo. Le azioni richiamabili dal nodo sono quattro:

- **Ricerca**: questa funzione permette al robot di generare una traiettoria casuale nello spazio di lavoro, con lo scopo di trovare una persona con cui iniziare un processo di interazione;
- **Avvicinamento**: richiamando questa funzione il robot, utilizzando i dati in suo possesso, genera una traiettoria di avvicinamento verso la persona appena trovata;
- **Navigazione**: genera una traiettoria di navigazione verso lo stand obiettivo, abilitando le funzioni di controllo per determinare se una persona lo sta seguendo e per cercarla in caso di smarrimento;
- **Abort**: questa funzione permette al robot di annullare ogni task di navigazione attivo. Viene utilizzata per permettere il blocco totale del robot da parte di un operatore.

Il nodo è stato implementato utilizzando due classi di ogetti. La prima definita nel file *Navigation.h*, definisce la classe *Navigation* in cui sono implementati i metodi precedentemente descritti, e le funzioni necessarie alla definizione di obiettivi sotto forma di coordinate sul piano cartesiano. Inoltre, sono disponibili i metodi che permettono di controllare lo stato del raggiungimento delle posizioni target e le funzioni che controllano i comportamenti di rilevamento e recovery delle persone con cui si sta interagendo.

La seconda classe, la *RobotInterface* ha lo scopo di semplificare il dialogo con i nodi che controllano il collo del robot, il sistema vocale, il sistema di riconoscimento facciale e il movimento del motore del kinect, in modo da semplificare la stesura del codice e il dialogo con i vari sottosistemi.

Il nodo è stato pensato in modo da essere altamente configurabile, in modo da abilitare o disabilitare le funzionalità utilizzabili dal nodo in esame. È possibile disabilitare i sistemi di voce, movimentazione del collo e kinect, e disabilitare l'addestramento del volto delle persone. Inoltre si possono impostare file differenti da quelli di default, in cui sono definiti le frasi da riprodurre, e i marker che identificano gli obiettivi di navigazione.

```
nh_.param<bool>("en_auto", find_user_, true);
nh_.param<bool>("en_neck", en_neck, true);
nh_.param<bool>("en_voice", en_voice, true);
nh_.param<bool>("en_train", en_train, true);
nh_.param<bool>("en_kinect", en_kinect, true);

nh_.param("marker_config", marker_config, ros::package::getPath("e2_config")
    +"/map_config/sim_marker_config.yaml");
nh_.param("speech_config", speech_config, ros::package::getPath("e2_config")
    +"/speak_config/speech_config.yaml");
```

Listing 4.3: Parametri Configurabili

Inoltre sempre in fase di inizializzazione vengono esportati diversi servizi che permettono di testare il corretto funzionamento del modulo senza dover avviare il modulo *e2_brain* che prevede lunghi tempi di interazione.

```
// Enable Services
abort_service_ = nh_.advertiseService(name+"/abort",&Navigation::
    abort_service,this);
goto_service_ = nh_.advertiseService(name+"/test_goto",&Navigation::
    goto_service,this);
talk_service_ = nh_.advertiseService(name+"/test_voice",&Navigation::
    talk_service,this);
neck_service_ = nh_.advertiseService(name+"/test_neck",&Navigation::
    neck_service,this);

find_user_service_ = nh_.advertiseService(name+"/find_user",&Navigation::
    find_user_service,this);
```

```
approach_user_service_= nh_.advertiseService(name+"/approach_user",&
    Navigation::approach_user_service,this);
navigate_target_service_ = nh_.advertiseService(name+"/navigate_target",&
    Navigation::navigate_target_service,this);
```

Listing 4.4: Servizi Navigazione

Per poter funzionare correttamente inoltre il nodo si sottoscrive a diversi topic. Il nodo utilizza messaggi provenienti dal topic */com*, che trasmette le informazioni delle persone rilevate da Kinect. In questo modo, il robot in fase di approccio ha sempre una posizione aggiornata e coerente della persona che deve raggiungere. Il topic */odom*, invece, fornisce continuamente informazioni sulla posizione del robot rispetto al frame */map*; in questo modo il robot riesce a salvare l'ultima posizione in cui si trovava a seguito del rilevamento di una persona. I topic */cmd_vel* e */e2_sonar* vengono utilizzati per avere informazioni riguardo al tipo di movimento che sta compiendo il robot, ad esempio se sta ruotando sul posto, e per ricevere informazioni dai sonar, quest'ultimi indispensabili per avere dati riguardo alla posizione dell'utente durante la navigazione verso lo stand. Questi dati sono vitali per l'interazione in quanto durante la navigazione il robot utilizza il sensore Kinect orientato frontalmente per rilevare ostacoli durante il tragitto e si suppone che la persona segua il robot, affiancandolo da un lato dello stesso.

```
//     Suscribers
face_sub_= nh_.subscribe("/com", 1,&Navigation::face_callback,this);
odom_sub_= nh_.subscribe("/odom",10,&Navigation::odometry_callback,this);
vel_sub_= nh_.subscribe("/cmd_vel", 10,&Navigation::velocity_callback,this);
sonar_sub_= nh_.subscribe("/e2_sonar",1,&Navigation::sonar_callback,this);
```

Listing 4.5: Topic sottoscritti

Terminato il processo di inizializzazione il modulo resta in attesa di richieste da parte del nodo *e2_brain*. A seguito della ricezione di un task, viene effettuata una chiamata alla funzione **executeCB** che ha il compito di scegliere l'azione corretta da eseguire.

```
void executeCB(const e2_navigation::NavGoalConstPtr& msg)
{
...
 switch(goal_id_)
 {
  case 0:      // Abort current action
       ROS_DEBUG("["ROS_NODE_NAME"]:: Abort current action");
```

```
        nav->ActionReset();
        as_.setAborted();
        return;
        break;
 case 1:      // Find user to start interaction
        ROS_DEBUG("["ROS_NODE_NAME"]:: Looking for user");
        nav->LookingUser();
        break;
 case 2:      // Aproach user if he's still far from the robot
        ROS_DEBUG("["ROS_NODE_NAME"]:: Aproaching user");
        nav->ApproachUser();
        break;
 case 3: // Start Navigation for interested people
        ROS_DEBUG("["ROS_NODE_NAME"]:: Start navigation to target");
        nav->NavigateTarget();
        break;
}

ros::Rate rate(ROS_NODE_RATE);

// Start Navigation Controller
while(!g_request_shutdown && !nav->isActionCompleted() && !nav->
    isActionAborted())
{
 nav->ActionController();
 ros::spinOnce();
 rate.sleep();
}

result_.action_id = goal_id_;

if(nav->isActionCompleted())
        as_.setSucceeded(result_,"OK");
else
        as_.setAborted(result_,"FAILED");
        nav->ActionReset();
}
```

Listing 4.6: Funzione Gestione task navigazione

La funzione *Navigation::ActionController()* è il cuore del modulo, e viene utilizzata per espletare il task richiesto. Questa viene eseguita ciclicamente fino al completamento dell'attività, controllando al termine di ogni esecuzione, la posizione del robot per stabilire il completamento dell'attività.

4.5.1 Ricerca e Avvicinamento

La prima tipologia di azioni attuabili dal nodo consiste nella ricerca di una persona nello spazio di lavoro. Il robot genera una traiettoria causale utilizzando i punti di via definiti all'interno dei file del pacchetto *e2_config*, presenti nella cartella *map_config*.

```
Marker:
  id: 0
  name: base
  position: [-1.018, -1.957, 0]
  orientation: [0.0, 1.0]

Marker:
  id: 1
  name: target
  position: [11.2477, -5.457, 0]
  orientation: [0.013, 0.999]

Marker:
  id: 2
  name: random_1
  position: [3.352, 0.853, 0]
  orientation: [0.723, 0.69]
  ....
```

Come si vede nel codice riportato, il file definisce differenti punti di via, tramite il formato *YAML*. I primi due punti identificano, rispettivamente, il punto dal quale il robot inizia il suo processo di interazione, e lo stand obiettivo che dovrà raggiungere. I marker successivi vengono richiamati per generare traiettorie casuali nell'ambiente.

Una volta generata la traiettoria, il robot si pone in attesa di ricevere informazioni dal nodo *openni-tracker*, il quale restituisce un frame che identifica il volto e il corpo di una persona rilevata nell'ambiente. Questi dati vengono utilizzati per generare una traiettoria e portarsi ad una distanza di interazione valutata intorno al metro di distanza dall'interlocutore.

4.5.2 Navigazione Stand Obiettivo

Una volta rilevata una persona interessata il modulo procede alla navigazione verso lo stand obiettivo. Viene chiamata la funzione *train_user()* che effettua l'addestramento del volto della persona da condurre; questa

fase verrà descritta nel capitolo successivo. Una volta terminato il processo il robot procede come segue. Viene dapprima generata una traiettoria verso lo stand obiettivo mediante la funzione *nav_goto(target_name_)* e vengono abilitati i servizi che permettono di tener traccia della posizione della persona rispetto al robot.

L'estratto successivo mostra come viene attivato il comportamento di *userDetection* utilizzato quando il robot perde il contatto con una persona.

```
if(path_planned_ && !rotating && navigate_target)
{
  // Left Data
  if(guest_user_info_.user_left )
  {
        if((msg->sonar6 > 0 || msg->sonar5 > 0 || msg->sonar4 > 0) && ((msg
            ->sonar6 < USER_SONAR_DISTANCE || msg->sonar5 <
            USER_SONAR_DISTANCE || msg->sonar4 < USER_SONAR_DISTANCE)) )
        {
          float diff_ = abs(msg->sonar6 - msg->sonar4 );

          if(diff_ < 5)
          {
                ROS_INFO("[Navigation::Sonar]:: Fake data!!!! Maybe a Wall.")
                    ;
                guest_user_info_.user_lost = true;
                init_detect_time = ros::Time::now();
          }
          else
          {
                ROS_INFO("[Navigation::Sonar]:: Utente a sinistra");

                guest_user_info_.user_lost = false;
                init_detect_time = ros::Time::now();
          }

        }
        else
        {
          ROS_ERROR("[Navigation::Sonar]:: User Lost by LEFT sonar");
          guest_user_info_.user_lost = true;
        }
  }
  ...
```

La funzione riceve costantemente informazioni dal sonar valutando le distanze percepite. Si considera che la persona si trovi accanto al

robot quando la distanza misurata risulta inferiore ai 45 cm. Inoltre è stato introdotto un meccanismo che faciliti il rilevamento di falsi positivi dovuti alla presenza di muri. L'idea di fondo è che i sonar non sono in grado di percepire contemporaneamente la persona a causa della loro posizione e dell'angolo di ricezione. Per questo motivo quando tutti e tre i sensori rilevano la stessa distanza, con un margine di errore di 5cm, sarà molto probabile che questa misurazione sia dovuta alla presenza di un muro.

Quando il robot rileva la mancanza della persona, dovuta alla mancata percezione della persona in un tempo di 3 secondi, viene lanciato il meccanismo di *UserDetection* descritto dalla funzione *Navigation::user_detectTimer* che permette al robot di ritrovare la persona.

```
ROS_INFO("[Navigation]:: Checking user presence.");

// First Stop every robot action
irobot_->cancell_all_goal();

// Remove current navigation goal
path_planned_=false;
user_recognized_=false;

ros::Time init_detection = ros::Time::now();
ros::Duration timeout(30.0);

while((ros::Time::now() - init_detection < timeout) && !user_recognized_)
{
	if(!rotating)
		if(guest_user_info_.user_left)
			irobot_->base_rotate(const_cast<char *>("LEFT"));
		else
			irobot_->base_rotate(const_cast<char *>("RIGHT"));

	//Check user presence
	user_detect(guest_name_);

	ros::spinOnce();
	r_.sleep();
}

irobot_->base_stop();
```

Inizialmente il robot arresta i vari task in esecuzione sul robot, fermandosi nella posizione in cui si trova. Successivamente il robot inizia

una rotazione rispetto alla posizione in cui si trovava la persona prima di essere persa. La funzione *user_detect* richiede al nodo di *face-recognition* di identificare la persona che stava accompagnando verso lo stand. Il processo termina se la persona viene identificata o si supera un timeout di 30 secondi.

Se la persona viene identificata viene lanciata la funzione *Navigation::user_wait* che pone il robot in una situazione di attesa. Questa ha il compito di richiamare la persona attraverso l'utilizzo della voce per notificare la posizione del robot, se la persona si trova ad una distanza superiore ai 2 metri e mezzo dal robot. Se dopo un intervallo di tempo il robot percepisce ancora una distanza elevata dalla persona, questi si avvicina a lei e la invita a seguirlo prima di poter riprendere la sua navigazione verso lo stand obiettivo.

Se invece la funzione di *userDetection* non rileva la persona, il robot esegue un altro comportamento, quello di *Backtracking*, che permette al robot di tornare all'ultima posizione in cui era stata rilevata visivamente la persona e viene generata una traiettoria fino a quel punto, interrogando il nodo di *face_recognition*. Se il robot arriva alla posizione senza trovare la persona, l'azione viene annullata in quanto si considera la persona scomparsa ed il robot comunica il suo fallimento al nodo *e2_brain* che in seguito deciderà l'azione successiva da compiere.

Capitolo 5

Sistema di Visione

"Il vero viaggio di scoperta non consiste nel cercare nuove terre, ma nell'avere nuovi occhi. "

Marcel Proust

Uno degli obiettivi primari della Computer Vision consiste nell'identificazione di oggetti e corpi all'interno di immagini digitali bidimensionali o 3D. Un sistema di visione artificiale è costituito dall'integrazione di componenti ottiche, elettroniche e meccaniche che permettono di acquisire, registrare ed elaborare immagini sia nello spettro della luce visibile, sia al di fuori di essa. Il risultato dell'elaborazione è il riconoscimento di determinate caratteristiche dell'immagine per varie finalità di controllo, classificazione, selezione e via dicendo.

Il riconoscimento di oggetti in tempo reale è quindi uno dei requisiti fondamentali per numerose applicazioni robotiche e, come in questo caso, lo sviluppo del modulo di visione è stato un passo fondamentale per gli scopi del presente elaborato. Il sistema sviluppato si affianca a quello precedentemente realizzato estendendo le funzionalità del robot, dotandolo della capacità di rilevare volti e di memorizzarli per poterli poi riconoscere nell'arco del processo di interazione.

Il capitolo è articolato descrivendo come vengono individuati i volti tramite classificazione, per poi focalizzare l'attenzione sul modulo *face-recognition* che implementa le funzionalità di rilevamento e riconoscimento.

5.1 Rilevamento volti

Il riconoscimento facciale è una tecnica di intelligenza artificiale, utilizzata in biometria per identificare o verificare l'identità di una persona a partire da una o più immagini che la ritraggono.

Tipicamente, l'identificazione avviene mediante tecniche di elaborazione digitale delle immagini, ignorando tutto quello che non rappresenta una volto, come edifici, oggetti o corpi estranei, che vengono solitamente considerati come sfondo. Si può affermare che si tratta di un riconoscimento di pattern, dove il pattern in questione è il volto umano. Per facilitare l'individuazione di una faccia, i primi sistemi tenevano conto di caratteristiche topologiche quali la presenza di due occhi, un naso ed una bocca. Tuttavia con i recenti sviluppi nel campo, è ora possibile riconoscere una persona anche se questa ha il viso ruotato o comunque non in posizione frontale, utilizzando tecniche più raffinate ed algoritmi più robusti.

Per questo progetto è stato utilizzato un classificatore Haar-like Cascade Viola and Jones [28]. Il punto di forza di questo sistema risiede in un algoritmo robusto ed estremamente rapido per il rilevamento di oggetti, e nella possibilità di poter essere applicato facilmente per individuare volti umani. Il modello si presta bene per lo scopo data l'estrema velocità dettata dalla frequenza di ricezione di immagini, e per l'utilizzo di immagini definite sulla scala di grigio, tutte caratteristiche tipiche del sensore Kinect. L'algoritmo garantisce un tasso di errore molto basso nel rilevamento, circa 1% di falsi negativi e il 40% di falsi positivi utilizzando un classificatore costruito utilizzando semplici Haar-like features.

Il sistema di *face-detection*, è composto da una cascata di classificatori detti stage. La cascata di classificatori riceve una sottofinestra dell'immagine di partenza e restituisce come output il risultato della classificazione, indicando se la porzione in esame è un volto umano. Gli stage sono posti in sequenza e in ordine di selettività crescente. Una finestra per essere classificata come contenente un volto, deve superare con successo tutti i vari stage.

La procedura di rilevamento classifica le immagini utilizzando delle

feature, il cui valore viene calcolato come la differenza tra la somma dei pixel all'interno delle regioni di due rettangoli, come mostrato in Figura 5.1. I rettangoli hanno la stessa forma e dimensione e possono essere orizzontalmente o verticalmente adiacenti. L'elaborazione di questi valori genera un elevato insieme di dati, basti pensare che utilizzando una finestratura di 24x24 pixel per l'elaborazione, è possibile estrapolare un numero di 180,000 feature dalla porzione in esame.

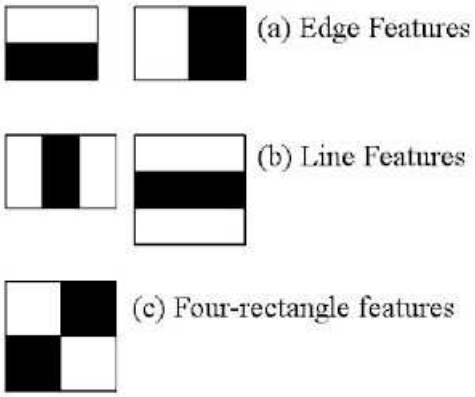


Figura 5.1: Features utilizzate nel processo di classificazione

Tuttavia non tutte le feature calcolate sono rilevanti ai fini dell'analisi. Ad esempio esaminando la Figura 5.2, si può evincere quali feature contribuiscano al rilevamento di pattern utili. La prima feature in esame si focalizza sulla proprietà che la regione degli occhi sia più scura di quella del naso e mento, mentre la seconda evidenzia che gli occhi sono più scuri rispetto al collegamento stabilito dal naso. Ma applicando la stessa finestra sul mento o ad altre regioni del volto, non si ottengono informazioni rilevanti. Per scartare quindi tutte queste feature, viene applicato un algoritmo di machine learning, AdaBoost [19].

Questo algoritmo di boosting permette di incrementare notevolamente la precisione di un algoritmo in fase di apprendimento, il cui obiettivo è quello di generare una serie di classificatori con scarse prestazioni, per poi combinarli in modo da avere un classificatore più robursto.

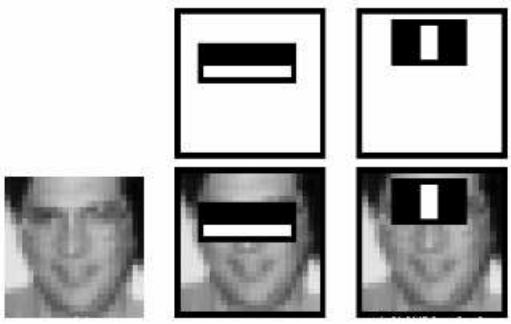


Figura 5.2: Applicazione di diverse feature ad un'immagine.

In questo modo il numero finale di applicazioni delle feature risulta drasticamente ridotto, circa 6000, partendo dalle 180,000 precedentemente identificate. Quindi una volta ottenuta un'immagine viene applicato questo insieme ridotto di feature per condurre l'analisi. Per ridurre ulteriormente il tempo di elaborazione, viene fatta un'ulteriore semplificazione. Se una regione dell'immagine non è un volto non verrà più processata, focalizzando quindi l'attenzione su altre regioni dove è più probabile che si trovi un viso, riducendo così il numero di elaborazioni.

Per questo è stato introdotto il concetto di Cascade of Classifiers. Invece di applicare tutte le 6000 feature in una finestra, queste vengono raggruppate in differenti passi, che vengono applicati in cascata: se l'applicazione di un set di feature fallisce al primo stadio, la porzione di immagine in esame viene scartata e il processo continua senza considerare le restanti feature nella porzione di finestra in esame. Se invece questa viene accettata, si procede allo stage successivo fin quando tutte le feature non sono applicate correttamente, e la porzione considerata viene rilevata come un volto.

Nel progetto sviluppato, il classificatore è implementato utilizzando la libreria OpenCV [3] (Open Source Computer Vision Library), un framework open source per la computer vision e il machine learning che implementa nativamente questo genere di classificazione.

5.2 Il modulo face-recognition

Il modulo *face-recognition*, come descritto precedentemente, ha il compito di rilevare i volti delle persone, e di confrontarli con quelli memorizzati all'inizio del processo di interazione con una persona. Il nodo è stato implementato per gestire due azioni principali: memorizzare il volto di una persona e riconoscerlo all'interno di una immagine.

Le chiamate al nodo vengono effettuate tramite la definizione di goal utilizzando l'interfaccia actionlib. Di seguito viene mostrata la definizione del *FaceRecognition.action*.

```
#goal definition
uint8 order_id
string order_argument
---
#result definition
uint8 order_id
float32[] angle
float32[] distance
string[] names
float32[] confidence
---
#feedback
uint8 order_id
float32[] angle
float32[] distance
string[] names
float32[] confidence
```

Il goal è definito da un valore numerico, definito dalla variabile *order_id* che identifica le seguenti azioni.

1. **Riconoscimento volto** (cod. 0 e 1): Il nodo può riconoscere il volto di una persona singolarmente su un singolo frame preso in esame o continuativamente aggiornando continuamente le informazioni provenienti da Kinect.

2. **Acquisizione volto** (cod. 2): Permette di salvare immagini del volto della persona che verranno analizzate per effettuare il riconoscimento.

3. **Estrazione Feature** (cod. 3): Permette di effettuare lo studio dei volti per estrapolare le informazioni e salvarle nel database.

4. **Terminazione** (cod. 4): Permette di terminare ogni operazione in corso e di uscire dal nodo.

Il parametro *order_argument* viene usato per associare un nome alla fase di rilevamento e memorizzazione che viene associato ai dati nel database.

Il nodo inoltre fornisce risposte sotto forma di messaggi che riportano informazioni utili alla posizione della persona rispetto al centro dalla camera. Le informazioni che vengono inviate come risposta sono l'identificativo dell'azione in esame, *order_argument*, l'angolo e la distanza dal centro ottico del sensore Kinect (*angle*,*distance*), il nome della persona trovata *names* e il valore di confidenza del matching.

Il nodo riceve le informazioni real-time dal sensore Kinect sottoscrivendosi ai topic messi a disposizione dal sistema ROS tramite i driver *Openni*. I topic di interesse sono quelli relativi al sensore RGB e quelli del sensore di profondità.

```
image_topic_rgb_("input_rgb"),
image_topic_depth_("input_depth"),
rgb_image_sub_( nh_ , image_topic_rgb_, 10 ),
depth_image_sub_( nh_ , image_topic_depth_, 10 ),
sync( MySyncPolicy( 10 ), rgb_image_sub_, depth_image_sub_ ),
```

Inoltre è necessario sincronizzare la ricezione dei due stream di immagini utilizzando la libreria *message_filters*. In questo modo la funzione *FaceRecognition::imageCB*, verrà richiamata utilizzando entrambi gli stream di immagini sincronizzate.

```
sync.registerCallback( boost::bind( &FaceRecognition::imageCB, this, _1, _2
    ) );
```

La funzione effettua il processo di rilevamento solo a seguito della ricezione del goal corretto.

Indipendentemente dal fatto che si voglia riconoscere o salvare un volto, la prima azione da compiere è quella di rilevare i volti nell'immagine. L'immagine ricevuta dal kinect viene quindi trasformata in scala di grigio e viene applicata la funzione *detectFaceInImage* che rileva i volti in un'immagine. La funzione riceve due parametri, l'immagine in

esame e il classificatore Haar cascade, le cui informazioni sono salvate nel file *haarcascade_frontalface_alt.xml* presente all'interno del nodo in esame.

```
const CvSize minFeatureSize = cvSize(20, 20);
const int flags = CV_HAAR_FIND_BIGGEST_OBJECT | CV_HAAR_DO_ROUGH_SEARCH;
const float search_scale_factor = 1.1f;
....

// Detect faces.
rects = cvHaarDetectObjects(detectImg, (CvHaarClassifierCascade*) cascade,
    storage, search_scale_factor, 4, flags, minFeatureSize);
```

All'interno della funzione sono evidenziati i parametri con cui verrà lanciato il processo di detection. La variabile *minFeatureSize* definisce la dimensione della porzione su cui applicare le varie feature per l'identificazione del volto. Viene usata una dimensione di 20x20 pixel. La variabile flags identifica quale tipologia di oggetti devono essere rilevati. Il valore HAAR_FIND_BIGGEST_OBJECT rileva gli oggetti più grandi nell'immagine, e insieme al parametro HAAR_DO_ROUGH_SEARCH costringe la funzione a non cercare candidati di dimensioni inferiori quando è stato trovato l'oggetto. Questi parametri consentono di velocizzare di molto l'analisi dell'immagine. Infine la variabile *scale* determina quanto deve essere scalata la finestra di ricerca tra scansioni successive. Il valore 1.1f si traduce in un incremento del 10% per la dimensione della finestra. Definiti tutti i parametri, viene lanciata la funzione *cvHaarDetectObjects* che restituisce una set di rettangoli le cui coordinate (in pixel) identificano la porzione in cui è presente un volto.

5.2.1 Memorizzazione Volto

Una volta rilevato un volto e se l'azione corrente è di memorizzarlo, il nodo procede al salvataggio di un numero di immagini, definite in fase di inizializzazione.

```
cvSaveImage(cstr, equalizedImg, NULL);
```

La funzione *cvSaveImage*, effettua il salvataggio della porzione di immagine del volto definita dalla variabile *cvSaveImage* nella cartella *data* presente nel pacchetto in esame, identificata dalla variabile *cstr*. I nomi delle immagini vengono inoltre salvati all'interno del file *train.txt*,

utilizzando il parametro ricevuto tramite la definizione del goal. Vengono salvate un numero di 25 immagini nel processo, soglia che si è dimostrata ampiamente sufficiente nella fase di riconoscimento.

Figura 5.3: Processo di addestramento

Una volta salvate le immagini viene effettuato un processo di PCA (Principal Component Analisys), una tecnica statistica, per trovare pattern con dimensionalità elevata. Questa tecnica permette di identificare pattern nei dati e li esprime mettendo in risalto le loro differenze e similitudini. Il PCA rappresenta i pixel dell'immagine nello spazio delle coordinate definito dalla base ortonormale, formata da alcuni degli autovettori della matrice di covarianza. Tale tecnica sfrutta il fatto che i template in esame, cioè le facce, sono altamente correlate e quindi molte coordinate sono poco significative. Gli autovettori che definiscono il sottospazio dei volti, vengono chiamati eigenfaces Turk and Pentland [27] e le immagini possono essere definite come combinazione lineare di questi ultimi.

Mediante l'utilizzo della funzione *cvEigenDecomposite* poi viene effettuata la riduzione di dimensionalità delle immagini da salvare, ottenendo un feature vector che viene salvato in un database, che verrà usato per le operazioni di riconoscimento future.

```
...
// Analisi immagini con PCA
PCA();

// Proietta le immagini da allenare nel sottospazio PC
projectedTrainFaceMat = cvCreateMat(nTrainFaces, nEigens, CV_32FC1);
offset = projectedTrainFaceMat->step / sizeof(float);

for (i = 0; i < nTrainFaces; i++)
```

```
{
        //int offset = i * nEigens;
        cvEigenDecomposite(faceImgArr[i], nEigens, eigenVectArr, 0, 0,
            pAvgTrainImg,projectedTrainFaceMat->data.fl + i * offset);
}

// Salva i dati appresi in un file xml
storeTrainingData();
...
```

Una volta condotta l'analisi, le informazioni vengono salvate su un database in formato xml, abilitando il riconoscimento del volto appena appreso.

5.2.2 Riconoscimento Volto

Il riconoscimento dei volti viene eseguito effettuando due step principali. Inizialmente viene estratto il feature vector associato all'immagine in esame, nel sottospazio dei volti identificati. Successivamente il vettore estrapolato sarà usato per effettuare il confronto di pattern simili a quelli presenti nel database creato nel paragrafo precedente.

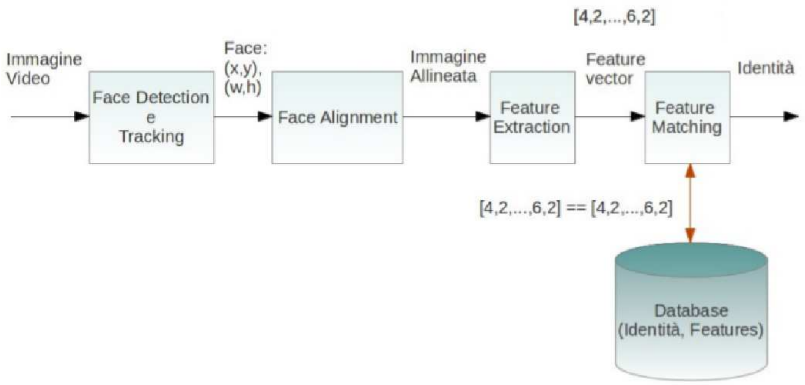


Figura 5.4: Processo di rilevamento

Nel primo step viene quindi proiettata l'immagine nel sottospazio dei volti, dopo averla centrata rispetto alla mean face. Il secondo passo consiste invece nel trovare l'immagine del training set che meglio approssima la sottofinestra in esame, e prende il nome di Nearest Neighbor matching. La soluzione più semplice, implementata nella funzione *findNearestNeighbor*, consiste nel valutare la distanza euclidea per misurare la differenza tra due feature vectors.

```
....
projectedTestFace = (float *) cvAlloc(frl.nEigens * sizeof(float));
cvEigenDecomposite(equalizedImg, frl.nEigens, frl.eigenVectArr, 0, 0, frl.
    pAvgTrainImg, projectedTestFace);

// Rileva il volto
iNearest = frl.findNearestNeighbor(projectedTestFace, &confidence);
nearest = frl.trainPersonNumMat->data.i[iNearest];
```

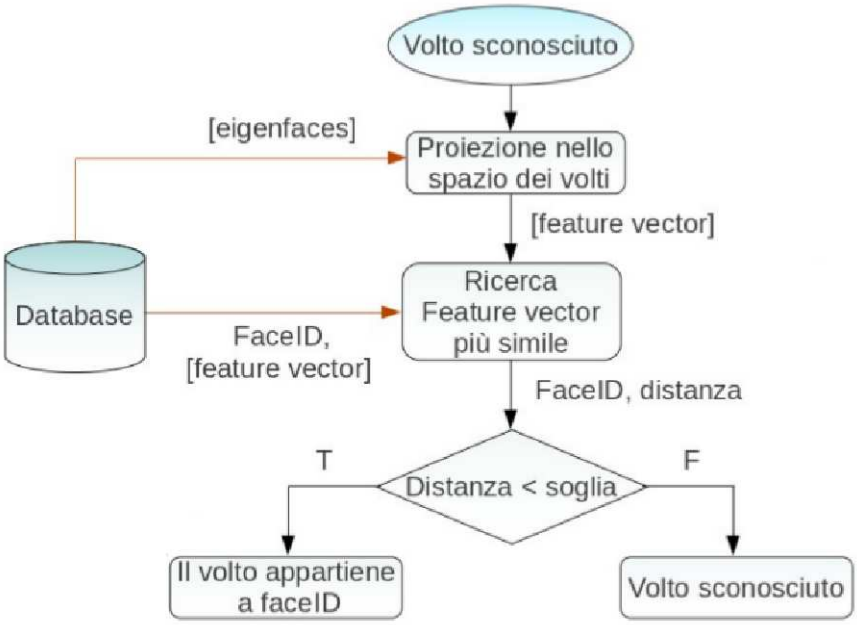


Figura 5.5: Processo di rilevamento

È possibile utilizzare anche un altro metodo, la distanza di Mahalanobis, che funziona meglio in quanto tiene in considerazione il fatto che, quando si proietta l'immagine di un volto nel sottospazio, ottenuto tramite PCA, ogni dimensione viene compressa o espansa a seconda dell'autovalore associato.

```
#ifdef USE_MAHALANOBIS_DISTANCE
  distSq += d_i*d_i / eigenValMat->data.fl[i]; // Mahalanobis distance
#else
  distSq += d_i * d_i; // Euclidean distance.
#endif
```

Terminato il processo, se la confidenza del risultato è sufficientemente elevata verrà generato un messaggio come mostrato nel dettaglio implementativo successivo, in cui sono riportati: la soglia di confidenza valutata dal processo di matching, la distanza dalla persona, e l'angolo

rispetto al centro della camera.

```
result_.names.push_back(frl.personNames[nearest - 1].c_str());
result_.confidence.push_back(confidence);
result_.angle.push_back(angle);
result_.distance.push_back(distance);
as_.setSucceeded(result_);
```

Queste ultime due informazioni vengono valutate rispettivamente, applicando il rettangolo del volto identificato sullo stream di profondità del Kinect e utilizzando la misurazione del centro dell'immagine; l'angolo invece viene calcolato attraverso una semplice interpolazione lineare: viene calcolata la distanza dal centro dell'immagine rispetto all'asse x e questa misurazione viene moltiplicata per la formula:

$$Ka = \frac{57}{(640^2 + 480^2)^{\frac{1}{2}}} \tag{5.1}$$

la quale esprime il rapporto tra l'angolo e i pixel del sensore Kinect. Nonostante il metodo non sia estremamente preciso in quanto l'angolo di vista orizzontale di 57° del sensore è frutto di una approssimazione e varia leggermente da dispositivo a dispositivo, un Kinect ben tarato fornisce buoni risultati con questa semplice applicazione.

Capitolo 6

Processo di Interazione

"Nessuno comprende l'altro. Siamo, come ha detto il poeta, isole nel mare della vita; tra noi si inserisce il mare che ci limita e separa. Per quanto una persona si sforzi di sapere chi sia l'altra persona, non riuscirà a sapere niente se non quello che la parola dice – ombra informe sul suolo della sua possibilità di intendere."

Fernando Pessoa, Il libro dell'inquietudine.

Come descritto nel corso del Capitolo 2, l'importanza di progettare comportamenti quanto più possibili human-like, determinano la buona riuscita di un robot sociale capace di interfacciarsi con un essere umano. Affinché l'interlocutore risulti realmente coinvolto nell'interazione è indispensabile che il robot comunichi una propria intenzionalità, ossia riesca a far credere alla persona di avere opinioni ed intenzioni. Inoltre, la capacità del robot di recepire comandi attraverso il parlato ne aumenta notevolmente il grado di partecipazione, dando all'interlocutore, anche se solo apparentemente, l'idea di comunicare con un essere senziente.

Il capitolo si articola descrivendo il modulo *e2_brain* che è stato oggetto di modifiche rispetto alla precedente realizzazione, in modo da aumentarne le potenzialità, per poi descrivere i moduli che intervengono nelle varie attività del robot e che si adoperano per aumentare il livello di coinvolgimento e interazione con le persone.

6.1 Il modulo e2_brain

Il modulo *e2_brain* si occupa di gestire le fasi principali del processo di interazione con l'utente. Il nodo è stato rivisitato e corretto, implementando nuove funzionalità. Il processo di interazione è articolato seguendo quattro passi necessari per un'interazione con una persona:

- **Ricerca**: Il primo step consiste nella ricerca di una persona nell'ambiente con cui iniziare un'interazione. Il robot inizia ad esplorare lo spazio di lavoro fin quando Kinect non identifica un volto umano, ricevendo i dati sulla sua posizione;
- **Avvicinamento**: il passo successivo consiste nell'avvicinarsi alla persona identificata, fino ad una distanza di interazione di un metro;
- **Valutazione interesse**: in questa fase il robot inizia a dialogare con la persona cercando di carpire la sua risposta emotiva e grado di interesse, mostrato a seguito di una serie di affermazioni del robot. Questa parte e' stata sviluppata in una tesi precedente e per maggiori informazioni sul processo di valutazione si rimanda a Mandelli and Zamponi [23];
- **Navigazione**: una volta stabilito che la persona è interessata il robot procede ad accompagnarla verso lo stand obiettivo, come descritto nel capitolo relativo alla navigazione.

Il nodo ha visto una riscrittura in funzione della nuova architettura *catkin* presente nelle nuove versioni del meta sistema ROS. Questo ha permesso di poter dialogare con tutti i vari sistemi del robot che sono stati realizzati con il paradigma *catkin*. Il nodo inoltre dialoga con tre componenti principali: il modulo di navigazione, il modulo per la movimentazione del kinect e con il sistema voce.

Una volta inizializzato, il sistema resta in attesa di comandi tramite il relativo servizio esportato dalla macchina.

```
//Services
ros::ServiceServer start_service = nh.advertiseService("e2_brain/start",
    start_callback);
```

```
ros::ServiceServer stop_service = nh.advertiseService("e2_brain/abort",
    stop_callback);
```

I servizi esportati in fase di inizializzazione attendono comandi da parte di una persona per iniziare il processo di interazione. Una volta avviato il robot, il servizio notifica ai vari sistemi le varie azioni da compiere mediante goal definiti tramite actionlib.

```
if(find_user)
{
  if(navHandlerFree)
  {
   ROS_ERROR("[e2_brain]:: Request to find a user sent at e2_navigation
       module...");
   navGoal.action_id = 1;
   navClient.sendGoal(navGoal, &navDoneCallback, &navActiveCallback, &
       navFeedbackCallback);
   navHandlerFree = false;
   navClient.waitForResult();
  }
}
```

Il nodo resta poi in attesa del completamento dell'operazione, ricevendo le notifiche dai sistemi chiamati, e gestendo le azioni da compiere. Parte dell'implementazione è mostrata nell'estratto seguente, che mostra come vengono gestite le chiamate terminate rispetto al modulo di navigazione.

```
void navDoneCallback(const actionlib::SimpleClientGoalState& state,const
    e2_navigation::NavResultConstPtr& result)
{
 navHandlerFree = true;
 ROS_INFO("[e2_brain]:: Navigation for task %d %s ",result->action_id,state.
     toString().c_str());

 if(state.toString() == "SUCCEEDED")
 {
  if(result->action_id == 1)
  {
    find_user=false;
       approach_user=true;
       check_user_interested=true;
  }
  else if(result->action_id == 3)
  {
       robot.action_completed++;
       initialize();
  }
```

```
 }
 else if(state.toString() == "ABORTED")
 {
  if(result->action_id == 3)
  {
        robot.action_aborted++;
        initialize();
  }
 }
}
```

La funzione *navDoneCallback* permette quindi di passare tra i vari step presenti nel modulo in questione, selezionando di volta in volta il passo successivo da compiere.

Inoltre, il modulo è stato dotato di una componente emotiva basata su una funzione che valuta le azioni che il robot riesce a compiere con successo e quelle che invece falliscono.

```
string temp_status = robot.status;

if(robot.action_aborted == robot.action_completed)
  robot.status = "normal_";
else if (robot.action_completed > robot.action_aborted)
{
  robot.status = "happy_";
  if((robot.action_completed - robot.action_aborted) > 3)
   robot.status = "good_";
}
else if (robot.action_aborted > robot.action_completed)
{
  robot.status = "angry_";
  if((robot.action_aborted - robot.action_completed) > 3)
   robot.status = "frustrated_";
}

if(strcmp(temp_status.c_str(),robot.status.c_str()) != 0)
{
  robot_talk(get_random_speech(robot.status));
  voiceClient->waitForResult();
}
```

Questa valutazione viene valutata ad ogni iterazione del nodo e il robot esprime differenti comportamenti definiti come frasi e movimenti del collo e del volto per manifestare il suo stato, a seguito del cambiamento di stato della funzione. Questi stati vengono poi eseguiti chia-

mando la funzione *robot_talk* che carica dal file di configurazione il testo da riprodurre e le azioni da compiere. Per una maggiore documentazione fare riferimento al modulo *e2_voice* che descrive l'implementazione del file di configurazione per questo genere di comportamenti.

Una delle novità proposte in questo elaborato consiste in un sistema di elaborazione del linguaggio naturale, che permette al robot di recepire comandi impartiti dalla persona per ampliare il livello di interazione con questa. Nonostante questa funzionalità sia ad un primo stadio di sviluppo, fornisce un'idea di quanto tale funzionalità sia importante per garantire un alto livello di interazione.

Il sistema si appoggia a CMU Sphinx [5], un riconoscitore del parlato, che mediante l'utilizzo di una grammatica riesce a riconoscere la voce percepita da un microfono con una buona approssimazione. Il sistema è stato testato con un semplice vocabolario di frasi. Per il corretto funzionamento è necessario disporre di un vocabolario di frasi, che verranno riconosciute, un modello che riconosca i fonemi delle parole e un modello statistico basato sulla frequenza con cui si presenteranno le varie parole.

Il sistema può essere adattato per riconoscere anche la lingua italiana, ma sarebbe necessario generare un modello specifico che richiederebbe diverse ore di sviluppo e non era previsto nei compiti del progetto. Questa grammatica può essere considerata semplicemente un input per futuri sviluppi.

```
ABORT  AH B AO R T
ARE    AA R
ARE(2) ER
DO     D UW
E_TWO  IY T UW
FEEL   F IY L
FIND   F AY N D
GO     G OW
HELL   HH EH L
HELLO  HH AH L OW
HELLO(2)      HH EH L OW
HOW    HH AW
JOKE   JH OW K
LAW    L AO
```

```
LAW(2) L AA
NEWS   N UW Z
NEWS(2) N Y UW Z
PEOPLE P IY P AH L
TELL   T EH L
THANK  TH AE NG K
YOU    Y UW
```

Listing 6.1: File dizionario

Il vocabolario implementato qui sopra è stato generato automaticamente da un parser grammaticale e prevede il riconoscimento di alcune parole con cui è possibile interagire con il robot. Accanto ad ogni parola sono associati i fonemi propri della parola che verranno usati dal riconoscitore sphinx per determinare la parola pronunciata.

Le informazioni provenienti dal microfono vengono quindi elaborate dal nodo *pocketphinx* e vengono inviate sul topic */recognizer/output* a cui si sottoscrive il modulo *e2_brain*; questi dati verranno successivamente processati dalla funzione *getVoiceCommands* che le elaborerà per determinare quale azione o risposta compiere.

```
ros::Subscriber subVoiceCommand = nh.subscribe("/recognizer/output", 10,
    getVoiceCommands);
```

Nel listato seguente è possibile avere un'idea di come vengano riconosciuti i comandi dal robot.

```
std::size_t found = command.data.find("e_two");

if (found!=std::string::npos)
{
 ROS_ERROR("[e2_brain::Voice]:: Ho ricevuto un comando !");

 found = command.data.find("find people");
 if (found!=std::string::npos)
 {
  ROS_ERROR("[e2_brain::Voice]:: Trova persone !");
                    robot_talk(get_speech_by_name("ack"));

  initialize();
  find_user = true;

  return;
 }
```

```
 found = command.data.find("tell joke");
 if (found!=std::string::npos)
 {
  ROS_ERROR("[e2_brain::Voice]:: Racconto una barzelletta!");
                    robot_talk(get_speech_by_name("ack"));
  voiceClient->waitForResult();
  robot_talk(get_random_speech(string("joke_")));
  voiceClient->waitForResult();
  return;
 }

...
 found = command.data.find("how do you feel");
 if (found!=std::string::npos)
 {
  ROS_ERROR("[e2_brain::Voice]:: Robot Status");

  robot_talk(get_random_speech(robot.status));
  voiceClient->waitForResult();
  return;
 }
...
```

Per poter interagire con il robot è necessario pronunciare il suo nome prima del comando o della richiesta che si vuole eseguire. Ad esempio pronunciando "E-2? how do you feel ?", il robot identificherà la richiesta e risponderà in base al suo stato emotivo interno come calcolato dalla funzione descritta nei paragrafi precedenti. Inoltre il robot è dotato di un ampia conoscenza di barzellette e aneddoti sulla robotica che possono essere richiamate mediante le richieste "tell joke" e "tell news".

Nonostante la semplice implementazione il sistema è un utile strumento per mostrare le capacità meccaniche ed interattive del robot attraverso un interfaccia basata sul linguaggio.

6.2 Il modulo e2_neck_controller

Il modulo *e2_neck_controller* è stato implementato per garantire il corretto dialogo con gli altri sistemi del robot e gestire le azioni della struttura superiore di E-2? durante l'interazione con le persone. Il modulo si occupa della messa in funzione dei servomotori presenti all'interno della testa e del collo di E-2?, al fine di dare al robot un'espressione coerente con la fase d'interazione in atto. La definizione delle azioni

disponibili di tale modulo è riportata all'interno del file Neck.action, contenuto all'interno della cartella action del modulo in analisi.

```
#goal definition
uint8  action
uint8  sub_action
---
#result definition
uint8  action_id
string result
---
#feedback
uint8  action_id
string status
```

Listing 6.2: Implementazione Neck.action

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

- **action**: questo parametro identifica i possibili set di azioni che possono essere richiamati; Gli insiemi di azioni disponibili sono quattro e sono cosi ripartiti: 1) identifica azioni per il movimento della faccia; 2) abilita le azioni disponibili per il collo del robot; 4) Identifica un insieme di azioni che permettono la configurazione real-time del collo, come velocità e accelerazione dei servomotori, nonché una funzione per permettere di scollegare in sicurezza il collo del robot.

- **sub_action**: identifica l'azione specifica relative all'insieme definito dal parametro *action*. Queste azioni verranno descritte in dettaglio successivamente.

I campi result e feedback sono sostanzialmente uguali con l'eccezione che il feedback viene costantemente inviato durante l'esecuzione di un'azione per controllarne lo stato, mentre il messaggio di result viene inviato una volta sola al completamento, e o al fallimento di una azione. Questi messaggi sono generati con la seguente descrizione:

- **action_id**: identifica il set dell'azione in corso.

- **result-status**: il parametro, rappresentato da una stringa, identifica lo stato dell'attività in corso o di quella terminata. Ad esempio a seguito di un'azione completata viene inviata la stringa *SUCCEDEED*, o in caso di fallimento la stringa *ABORTED*.

Il robot è quindi in grado di mostrare differenti espressioni facciali, attraverso l'attuazione dei servomotori. Le possibili espressioni sono state configurate utilizzando lo script interno alla scheda Mini Maestro Pololu che gestisce il movimento della parte superiore del robot. Le possibili espressioni mostrate sono elencate di seguito con riferimento al codice identificativo del parametro *sub_action*.

- **Standard (cod. 1)**: espressione neutrale del robot, utilizzata come posizione di partenza per i servomotori.
- **Happy (cod. 2)**: espressione utilizzata dal robot ogni qualvolta riesce in un task o riceve complimenti da una persona.
- **Angry (cod. 3)**: espressione utilizzata dal robot a seguito di un fallimento nel processo di interazione o quando il robot rileva di parlare con una persona sgarbata.
- **Interested (cod. 4)**: espressione utilizzata quando il robot interagisce con un interlocutore particolarmente interessato.
- **Invitation (cod. 5)**:espressione utilizzata nella fase di invito all'interlocutore a seguire il robot verso un particolare punto di interesse.
- **Start Speaking (cod. 6)**: Permette di attivare il movimento della bocca.
- **Stop Speaking (cod. 7)**: Permette di terminare il movimento della bocca.

Inoltre queste espressioni possono esser mostrate assieme all'emissione di frasi da parte del robot, come verrà spiegato nel modulo *e2_voice*. Tra i movimenti possibili nella sezione del robot è possibile attivare una serie di movimenti del collo, anche questi utilizzati durante la comunicazione con una persona e per manifestare particolari stati emotivi di E-2?.

(a) Normal *(b) Happy* *(c) Interested*

(d) Invite *(e) Angry* *(f) Annoyed*

Figura 6.1: Espressioni riproducibili su E-2?

- **Straight Neck (cod. 1)**: è la posizione base del collo, utilizzata durante la fase di navigazione. È importante far notare che per ottenere una corretta navigazione, il collo deve trovarsi in posizione eretta per non compromettere la validità dei dati ricevuti dal kinect, che verranno poi processati dal nodo *amcl*.
- **Invitation Left (cod. 2)**: il collo effettua una movimento diagonale all'indietro verso sinistra prima di ritornare in posizione eretta. Questo movimento, seguito da una frase sta ad indicare all'interlocutore l'invito del robot a seguirlo.
- **Invitation Right (cod. 3)**: effettua un movimento simmetrico al precedente, con la differenza del lato in cui è diretto. Il significato è il medesimo.
- **Give a Bow (cod. 4)**: il collo effettua un breve inchino, prima di rialzarsi in posizione eretta. Questo movimento è stato utilizzato al termine del processo di interazione per permettere al robot di manifestare la sua gratitudine ad averlo seguito.

- **Bend Forward (cod. 5)**: il collo viene inclinato in avanti. Questo rimane in questa posizione fin quando non viene richiesto di rialzarsi in posizione eretta.
- **Bend Back (cod. 6)**: permette il movimento all'indietro del collo. Questo rimane in questa posizione fin quando non viene richiesto di rialzarsi in posizione eretta.
- **Bend Left (cod. 7)**: permette di inclinare il collo del robot verso sinistra. Come per le precedenti, il robot resta in attesa in questa posizione.
- **Bend Right (cod. 8)**: come il precedente, con la differenza del movimento verso destra.

Inoltre gli script presenti sulla scheda Pololu sono stati modificati per permettere un movimento più naturale del collo. A causa della natura dei servomotori, si verificavano vistose oscillazioni meccaniche, dovute alle velocità di arrivo alle posizioni finali. Questo causava una collisione con il blocco meccanico presente al fine corsa. Questo problema è stato risolto utilizzando più set point nella risalita e nella discesa del movimento, adottando un andamento a rampa per le velocità dei servomotori, che si è tradotto in un movimento più fluido e naturale.

6.3 Il modulo e2_voice

Il modulo *e2_voice*, 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 *Voice.action*, contenuto nella cartella action del modulo in analisi.

```
#goal definition
uint8 action_id
string text
uint8 neck_action
uint8 face_action
---
#result definition
uint8 action_id
string result
---
```

```
#feedback
uint8 action_id
string status
```

Listing 6.3: Implementazione Voice.action

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

- **action_id**: questo parametro identifica l'insieme di azioni da richiamare. Questo consente di annullare (cod. 0) una azione corrente o di sintetizzare una frase ricevuta dal nodo (cod. 1).
- **text**: questo parametro identifica la stringa di testo che deve essere sintetizzata dal nodo in modo da essere convertita in un file audio e essere poi riprodotta dal nodo.
- **neck_action**: definisce una specifica azione che deve essere compiuta dal robot durante la riproduzione di una frase.
- **face_action**: identifica quale espressione facciale deve essere associata alla frase riprodotta dal robot.

Le altre specifiche del file *Voice.action*, riguardanti feedback e result sono le stesse descritte in precedenza per il nodo *e2_neck_controller*. Inoltre, il modulo voce utilizza il programma *pico2wave* per sintetizzare stringhe di testo in un file audio. Questo programma utilizza **SVOX**, un sintetizzatore vocale che garantisce ottime prestazioni ed è utilizzato nei principali sistemi di navigazione in commercio. A causa di alcune limitazioni la voce italiana è disponibile solo nella versione femminile, quindi, data la natura mascolina del robot E-2?, è stato implementato un filtro per modulare la voce risultante e, aumentando le basse frequenze, si è ottenuta una voce più consona al robot. Di seguito viene mostrata l'implementazione del comando usato per sintetizzare la voce di E-2?.

```
string command="pico2wave -l it-IT -w /tmp/e2.wav '"+msg->text+"' && play /
    tmp/e2.wav pitch -190 stretch 0.9 band 3000 500 treble 10 >/dev/null
    2>&1";
```

Listing 6.4: Comando sintetizzatore vocale

Inoltre, il modulo voce gestisce automaticamente la movimentazione della bocca del robot, in modo da avere un movimento sincrono tra la riproduzione della voce e l'attuazione del controllo dei servomotori della faccia. Infine, come descritto in precedenza, è possibile associare alla frase riprodotta una particolare espressione facciale o movimento della faccia. Questo è stato ottenuto nella definizione del vocabolario di E-2? tramite un file formattato tramite **YAML**, una libreria che permette di avere una buona standardizzazione nella definizione di un file. Queste azioni vengono espletate prima di iniziare a riprodurre il movimento della voce come si può osservare dall'estratto successivo.

```
// Add face expression
if(msg->face_action != 0 )
{
  n_goal.action=1;
  n_goal.sub_action=msg->face_action;
  ac_nc->sendGoal(n_goal);
  ac_nc->waitForResult();
}
// Add neck action
if(msg->neck_action != 0)
{
  n_goal.action=2;
  n_goal.sub_action=msg->neck_action;
  ac_nc->sendGoal(n_goal);
  ac_nc->waitForResult();
}

// Start Moving mouth
n_goal.action=1;
n_goal.sub_action=6;
ac_nc->sendGoal(n_goal);
....
```

Listing 6.5: Comando sintetizzatore vocale

Il robot è stato dotato di una serie di frasi che possono essere riprodotte durante la navigazione o a seguito di una richiesta della persona. Tra queste, è possibile trovare un'ampia gamma di barzellette, aneddoti sulla robotica, ed un insieme di frasi utilizzate durante i processi di interazione, facilmente estendibili. Queste sono definite nel file *speech_config* all'interno del pacchetto *e2_config*, nella cartella *speak_config*, di cui viene mostrata parte dell'implementazione.

```
....

#####################################################
# Jokes
#####################################################
Speech:
  id: joke_1
  text: "Squilla il telefono, un robot si avvicina e chiede: Che cosa parla?
      "
  neck_action: 0
  face_action: 2

Speech:
  id: joke_2
  text: "Sapete perch[U+FFFD] la neve cade a fiocchi e non a nodi ?
      Perch[U+FFFD] se cadesse a nodi farebbe molta pi[U+FFFD] fatica a
      sciogliersi!"
  neck_action: 6
  face_action: 2

Speech:
  id: joke_3
  text: "Va bene, va bene! Lo capisco che non sono accettato dal fatto che
      non sanguino."
  neck_action: 0
  face_action: 2

Speech:
  id: joke_4
  text: "Ero andato a comprare dei chiodi di garofano. Devo piantare la mia
      fidanzata e volevo dirglielo con i fiori."
  neck_action: 5
  face_action: 2

......
```

Listing 6.6: Frasi di E-2?

Capitolo 7

Verifiche Sperimentali

"L'esperienza è il tipo di insegnante più difficile. Prima ti fa l'esame, poi ti spiega la lezione. "

Oscar Wilde

In questa sezione verranno illustrati i risultati ottenuti mediante le prove svolte sul robot, al fine di valutare pregi e difetti del sistema di controllo e per determinare l'accuratezza dei modelli impiegati nella sua realizzazione.

Verranno quindi discussi e valutati i sistemi di visione e di navigazione, utilizzando come fonte di conoscenza l'esperienza maturata durante i vari test condotti.

7.1 Valutazione del Sistema di Navigazione

La valutazione del sistema di navigazione tiene in considerazione, l'affidabilità del sistema di localizzazione e la percentuale di fallimenti dovuti a collisioni con oggetti o ostacoli. Il robot riesce a fronteggiare senza problemi ostacoli frontali e a superarli senza troppe difficoltà, riuscendo a spostarsi senza problemi anche in spazi delle dimensioni pari al doppio della sua larghezza.

Il sistema di localizzazione, infine, garantisce un buon grado di approssimazione, nonostante il semplice utilizzo del sensore Kinect. Nelle prove effettuate, in un laboratorio del Politecnico, il sistema ha mostrato un tasso di errore prossimo allo zero, riuscendo a terminare il

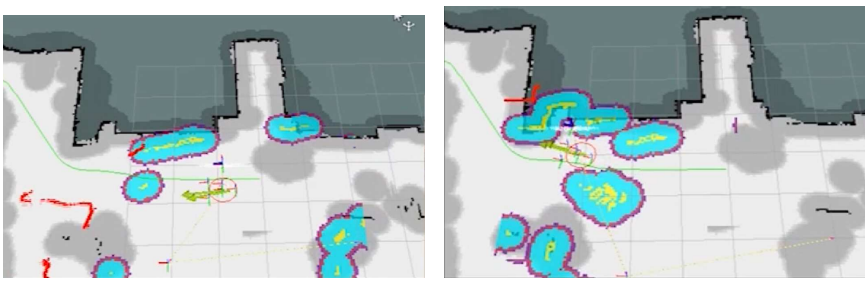


Figura 7.1: Vista degli ostacoli. rviz

percorso di navigazione senza collisioni. L'accuratezza del sistema di navigazione è difficile da stimare lungo tutto il tragitto. In una prova condotta in laboratorio, utilizzando distanze fisse si è riscontrato un errore medio di posizione di 4cm sulla componente lineare e inferiore ai 10° per la posa finale del robot. Questo errore tuttavia, viene compensato da AMCL che riesce a garantire che l'errore non aumenti nel tempo.

Alcune limitazioni sono state imposte dal pianificatore di movimento che viene utilizzato con una bassa frequenza e simulando brevi traiettorie di movimento per non sovraccaricare troppo l'elaboratore nelle situazioni di pieno carico operativo, in cui si potrebbero degradare le prestazioni. Inoltre, alcuni problemi si sono evidenziati nell'attuazione meccanica delle ruote, che vengono alimentate a massima potenza senza effettuare alcun controllo sulla velocità. Questo problema potrebbe essere risolto aggiungendo un controllo proporzionale direttamente nelle schede motori, riducendo le oscillazioni che si presentano sul collo dovuto a brusche accelerazioni e decellerazioni.

7.2 Valutazione del Sistema di Visione

L'obiettivo del sistema di visione, era di poter disporre di un metodo che permettesse al robot di riconoscere un volto umano in differenti contesti dinamici in cui il robot dovrà operare. Era necessario valutarne le prestazioni utilizzando differenti condizioni ambientali, per valutare la risposta del sistema. I test sono stati condotti con differenti fonti luminose per valutarne l'impatto e utilizzando differenti posture del volto per determinare il tasso di successo nelle fasi di detection.

Figura 7.2: Fase di rilevamento volto

La fase di tracking viene eseguita correttamente senza problemi e il robot è in grado di seguire lo spostamento di una persona senza difficoltà una volta identificatala e generato lo scheletro del corpo utilizzando il firmware del Kinect.

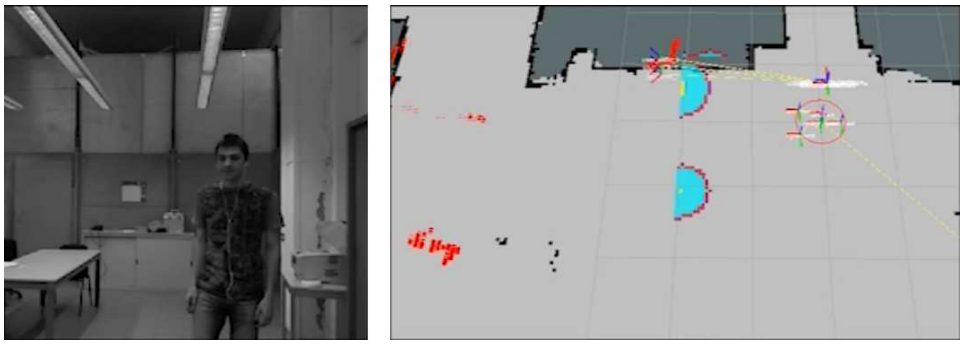


Figura 7.3: Sistema di tracking

7.3 Variazioni Luminose

Le variazioni dell'intensità luminosa incidono notevolmente sulla fase di addestramento del volto. Si è riscontrato un minor numero di successi se l'apprendimento del volto avviene in condizioni di illuminazione diverse da quelle in cui viene effettuata la detection.

Questo problema si è riscontrato, anche se con un tasso inferiore, nell'identificazione dei volti. Tuttavia, questi problemi si sono verificati in condizioni di illuminazione minime, che difficilmente possono presentarsi in contesti quali quelli in cui il robot sarà chiamato ad operare, ove si sono riscontrati falsi positivi con un tasso prossimo allo zero nella

maggior parte delle interazioni con il robot.

Figura 7.4: Processo di interazione di E-2?

7.4 Valutazioni generali

Valutare il processo di interazione generale è molto complesso, in quanto la logica del sistema dipende da molti moduli che devono collaborare tra di loro e quindi anche l'analisi deve essere valutata considerando ogni aspetto del problema.

Complessivamente tutti i sistemi riescono a collaborare e il sistema di navigazione proposto riesce ad adattarsi ai vari ambienti dopo una accurata costruzione della mappa. Il robot riesce poi ad effettuare l'intero processo di interazione non evidenziando problemi, se non dovuti a condizioni di luce proibitive. Quindi, nel complesso, il sistema è capace di gestire un'interazione completa e continuativa con le persone dimostrando di aver centrato gli obiettivi posti per questa tesi.

Capitolo 8

Conclusioni e Sviluppi futuri

"There is no real ending. It's just the place where you stop the story"

Frank Herbert

Lo scopo di questo lavoro era quello di creare un sistema di controllo autonomo per un robot da esibizione, tale E-2?, che fosse in grado di muoversi autonomamente in ambienti complessi e dinamici e che fosse in grado di instaurare un'interazione con le persone presenti, mirata a convincerle a seguirlo a uno stand specifico.

Il sistema realizzato ha dimostrato come, con il semplice utilizzo di un sensore Kinect, sia stato possibile ottenere un robot in grado di trovare persone in un ambiente esteso e complesso, di analizzare il loro stato attenzionale e di mantenere un buon grado di interazione con loro. Lo studio si è poi concentrato sulle azioni compiute dal robot, come espressioni facciali associate a particolari frasi espresse dalla macchina a seguito delle sue azioni o di quelle della persona.

Guardando a sviluppi futuri, i miglioramenti sono limitati semplicemente dalla fantasia dei "creativi" (programmatori umani). Una delle possibili migliorie potrebbe riguardare l'espansione del modulo *e2_brain* in merito all'interfaccia di elaborazione del linguaggio naturale per aumentare il numero di interazioni del robot. Ad esempio il robot potrebbe essere esteso per rispondere a comandi o quesiti della persona utilizzando una base di conoscenza più estesa affiancata da un parser linguistico per determinare grammaticalmente il tipo di azione da compiere, ad esempio se si tratta di un quesito o di un ordine. L'im-

plementazione attuale è generata su una grammatica molto semplice e riconosce comandi o domande specifiche. Uno sviluppo in questa direzione aumenterebbe notevolmente il grado di interazione con il robot, coinvolgendo maggiormente la persona, dando l'impressione di dialogare con un essere senziente.

In conclusione, il progetto concluso ha portato alla realizzazione di un robot in grado di svolgere il compito proposto, che si pone come un punto di partenza per lo sviluppo di robot sociali aventi l'obiettivo di riprodurre un'interazione human-like, in differenti contesti operativi.

Bibliografia

[1] URL `www.ros.org/actionlib`.

[2] URL `http://wiki.ros.org/navfn`.

[3] URL `opencv.org/`.

[4] URL `www.ros.org`.

[5] URL `http://cmusphinx.sourceforge.net/wiki/`.

[6] URL `http://airlab.ws.dei.polimi.it/index.php/Triskar2`.

[7] Tracy L Anderson and Max Donath. Animal behavior as a paradigm for developing robot autonomy. *Robotics and Autonomous Systems*, 6(1):145–168, 1990.

[8] Isaac Asimov. *I, Robot*. Mondadori, 1950.

[9] Andrea Bonarini, Matteo Matteucci, Martino Migliavacca, and Davide Rizzi. R2p: An open source modular architecture for rapid prototyping of robotics applications. In *Embedded Systems, Computational Intelligence and Telematics in Control*, number 1, pages 68–73, 2012.

[10] Cynthia Breazeal. Toward sociable robots. *Robotics and autonomous systems*, 42(3):167–175, 2003.

[11] Cynthia Breazeal and Brian Scassellati. A context-dependent attention system for a social robot. *rn*, 255:3, 1999.

[12] Allison Bruce, Illah Nourbakhsh, and Reid Simmons. The role of expressiveness and attention in human-robot interaction. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 4, pages 4138–4142. IEEE, 2002.

[13] Amedeo Cappelli, Emiliano Giovannetti, and KDD Laboratorio. L'interazione uomo-robot. Technical report, RoboCare Technical Reports, 2003.

[14] Giovanni Di Sirio. Chibios/rt homepage. *ChibiOS/RT free embedded RTOS*.

[15] Terrence Fong, Illah Nourbakhsh, and Kerstin Dautenhahn. A survey of socially interactive robots. *Robotics and autonomous systems*, 42(3):143–166, 2003.

[16] Tully Foote. tf: The transform library. In *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, pages 1–6. IEEE, 2013.

[17] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[18] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *AAAI/IAAI*, 1999:343–349, 1999.

[19] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, pages 23–37. Springer, 1995.

[20] Giuseppina Gini and Vincenzo Cagliotti. *Robotica*. 2007.

[21] Kenji Kaneko, Fumio Kanehiro, Mitsuharu Morisawa, Kanako Miura, Shinichiro Nakaoka, and Shuuji Kajita. Cybernetic human hrp-4c. In *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*, pages 7–14. IEEE, 2009.

[22] Cory D Kidd, Will Taggart, and Sherry Turkle. A sociable robot to encourage social interaction among the elderly. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 3972–3976. IEEE, 2006.

[23] Cristian Mandelli and Deborah Zamponi. Studio e realizzazione di un sistema d'interazione uomo-robot. 2012.

[24] Giorgio Metta, Giulio Sandini, David Vernon, Lorenzo Natale, and Francesco Nori. The icub humanoid robot: an open platform for

research in embodied cognition. In *Proceedings of the 8th workshop on performance metrics for intelligent systems*, pages 50–56. ACM, 2008.

[25] Alireza Fadaei Tehrani, Ali Mohammad Doosthosseini, Hamid Reza Moballegh, Peiman Amini, and Mohammad Mehdi DaneshPanah. A new odometry system to reduce asymmetric errors for omnidirectional mobile robots. In *RoboCup 2003: Robot Soccer World Cup VII*, pages 600–610. Springer, 2004.

[26] Ching-Chih Tsai, Li-Bin Jiang, Tai-Yu Wang, and Tung-Sheng Wang. Kinematics control of an omnidirectional mobile robot. In *Proceedings of 2005 CACS Automatic Control Conference*, pages 13–18, 2005.

[27] Matthew A Turk and Alex P Pentland. Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pages 586–591. IEEE, 1991.

[28] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

Appendice A

Documentazione del software

Il sistema è stato sviluppato inizialmente sul simulatore V-Rep, per poi spostare lo sviluppo sul calcolatore del robot sul quale è installato il sistema operativo Ubuntu Linux (14.04 LTS).

Per poter utilizzare il sistema correttamente è necessario installare una versione ROS (Robot Operating System), seguendo il processo descritto sul sito *www.ros.org*. Il sistema è stato sviluppato utilizzando la versione *Hydro*, anche se è stato correttamente testato sulla versione *Indigo* dello stesso.

Inoltre è necessario installare i seguenti pacchetti, se ancora non presenti:

- ros-hydro-navigation
- ros-hydro-openni-tracker
- ros-hydro-actionlib
- ros-hydro-pocketshinx
- ros-hydro-openni-launch
- pico2wave

Infine, per testare il funzionamento del robot usando gli ambienti creati è necessario installare il simulatore V-REP, seguendo le procedure descritte sul sito http://www.coppeliarobotics.com/

A.1 Accesso Remoto E-2?

Per accedere da remoto al robot è possibile sfruttare la connessione interna del laboratorio *airlab2* utilizzando la password interna.

L'accesso al sistema è ottenuto tramite connessione ssh con l'account *AIRLab2* e la solita password.

A.2 Compilazione sorgenti

L'intero progetto può essere scaricato dal repository git disponibile all'indirizzo *https://github.com/boottp/e2*.

Il progetto deve essere scaricato nella cartella *src* del proprio workspace catkin. Per farlo basta lanciare il seguente comando all'interno della cartella.

```
git clone https://github.com/boottp/e2
```

A.2.1 Compilazione ann library

Per compilare la libreria che elabora le informazioni basta digitare i seguenti comandi all'interno della cartella appena creata *e2*.

```
$ cd e2/ann/ann-build
$ make installed
$ cp ann/lib ../ -R
```

A.2.2 Compilazione Sistema

L'intero sistema è stato sviluppato con il nuovo sistema *catkin*, quindi la compilazione si effettua compilando il workspace.

Entrare nel proprio workspace catkin (*roscd*) e digitare il seguente comando

```
$ catkin_make
```

Una volta terminata la compilazione il sistema è pronto per l'utilizzo.

A.2.3 Creazione Ambiente di lavoro

La creazione della mappa dell'ambiente di lavoro deve essere effettuata prima di poter avviare il robot. Questa operazione è resa disponibile tramite lo script; collegare il joystick del robot e digitare

```
$ roslaunch e2_launch e2_robot_ambient_map.launch
```

Una volta avviato si aprirà il simulatore rviz, che mostra la creazione della mappa mentre si guida il robot nello spazio. Una volta ottenuta una mappa soddisfacente, salvarla con il comando

```
$ rosrun map_server map_saver -f mappa
```

Copiare la nuova mappa all'interno del pacchetto *e2_config*, nella cartella *map*. Creare un nuovo file di configurazione per gli stand obiettivo sulla falsa riga di quelli presenti nella cartella *map_config* dello stesso pacchetto. Infine aggiornare i riferimenti nello script di avvio *e2_robot.launch*

A.2.4 Avvio del Sistema

Il sistema può essere avviato richiamando lo script di avvio attraverso il comando

```
$ roslaunch e2_launch e2_robot.launch
```

Il comando appena digitato avvierà automaticamente tutti i nodi richiesti per il funzionamento dell'intero processo. Opzionalmente, è possibile avviare singolarmente ogni componente. Ogni sottosistema o dispositivo che può essere avviato è dotato di un proprio script di avvio. Per avviare singolarmente digitare

```
$ roslaunch e2_launch components/\textit{sistema}.launch
```

sostituendo *sistema* con lo script che si vuole caricare. Questo permette di controllare singolarmente ogni modulo del robot. Per maggiori informazioni sugli script disponibili, esaminare il contenuto del pacchetto *e2_launch* nello specifico la cartella *launch*.

A.2.5 Avvio processo interazione

Una volta che il robot è pronto e carico può essere avviato il processo di interazione attraverso il servizio *start* esportato dal nodo *e2_brain*.

```
$ rosservice call /e2_brain/start
```

Il robot inizierà a cercare autonomamente persone nell'ambiente con cui iniziare un processo di interazione.

Per terminare il processo si può usare il servizio *stop* esportato dallo stesso nodo.

```
$ rosservice call /e2_brain/stop
```

Inoltre è possibile testare singolarmente le funzionalità dei sistemi di navigazione mediante i servizi resi disponibili dal nodo. Di seguito vengono presentati i principali servizi esportati con relativa funzione di utilizzo per il nodo *e2-navigation*.

1. *test_goto* **param**: Questo servizio permette di far muovere il robot fino al punto obiettivo identificato dalla variabile **param** come definito nel file *marker_config*;
2. *abort*: Permette di terminare ogni attività presente nel nodo in esame;
3. *find_user*: Effettua la ricerca di una persona nell'ambiente, finché non ne viene identificata una;
4. *navigate_target*: Procede verso lo stand obiettivo effettuando le procedure di detection e backtracking;

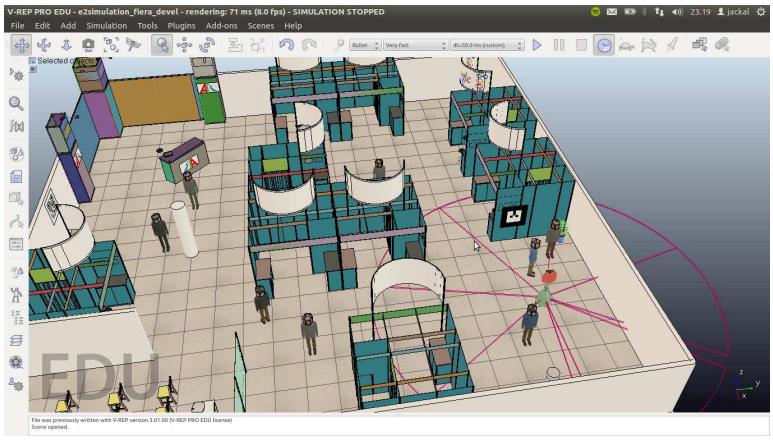


Figura A.1: Ambiente Simulazione

A.3 Simulatore

Il robot viene in dotazione con degli ambienti già configurati per il simulatore V-Rep. Per testare il funzionamento del robot avviare

il simulatore e caricare uno degli ambienti disponibili nella cartella *vrep_scenario*.

Si aprirà un ambiente come quello mostrato in Figura A.1; eseguire quindi il comando qui di seguito per avviare il controllo.

```
$ roslaunch e2_launch simulation_robot.launch
```

Una volta avviato si possono testare le funzionalità come descritto nel paragrafo precedente.

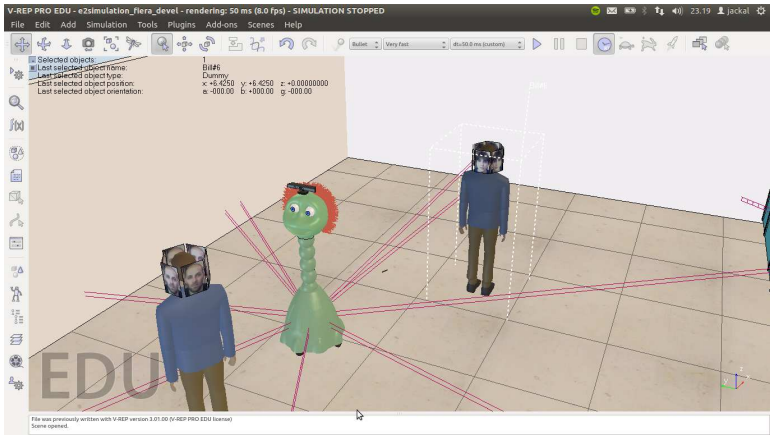


Figura A.2: Vista E-2?