

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Seconda Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

ESPERIENZE IN ROBOTICA CON LEGO
MINDSTORM E RESPECT

Elaborata nel corso di: Sistemi Distribuiti LA

Tesi di Laurea di:
MATTEO AMADUCCI

Relatore:
Prof. ANDREA OMICINI

Correlatore:
Prof. MATTEO CASADEI

ANNO ACCADEMICO 2008–2009
SESSIONE I

Ai miei famigliari...

Indice

Introduzione	iii
0.1 Scenario	iii
0.2 Obiettivi	iii
1 Intelligenza artificiale e robotica	1
1.1 Le leggi della robotica	2
1.2 Automi e robot	3
1.3 Uso contemporaneo dei robot	4
1.3.1 La robotica umanoide	5
1.3.2 Reti neurali	6
1.4 Linee di sviluppo della robotica	7
1.5 Componenti di un robot	9
1.6 Alcuni esempi	10
2 Sistema di controllo per un robot mobile	13
2.1 Requisiti	14
2.2 Livelli e strati	15
2.2.1 Livelli di competenza	15
2.2.2 Strati di controllo	17
2.3 Struttura degli strati	18
2.3.1 Comunicazione fra moduli	19
2.4 Livello 0	20
2.5 Livello 1	21
3 Lego Mindstorms	23
3.1 Panoramica sulla tecnologia NXT	24
4 Lego Mindstorm e ReSpect	27
4.1 Linguaggio di coordinazione	27
4.1.1 Tuple-Space come meta modello	28
4.1.2 Da Linda a ReSpect	29
4.2 Architettura basata su ReSpect per la Situatedness . .	33

4.3	Un MAS per il controllo di un robot mobile	35
4.3.1	Accesso alle risorse ambientali	39
4.3.2	Architettura del MAS basata su ReSpecT	40
4.3.3	Livello 0: evitare gli ostacoli	45
4.3.4	Livello 1: muoversi nell'ambiente	48
5	Esempio applicativo	51

Introduzione

0.1 Scenario

Fin dall'antichità, una delle ambizioni dell'uomo è stata quella di creare qualcosa di simile a se stesso, ovvero un'entità capace, da un lato, di sostituirlo nello svolgimento di compiti gravosi e, dall' altro, di riprodurre comportamenti intelligenti. I primi automi meccanici, realizzati intorno al 1700, avevano lo scopo di divertire la gente, erano poco più che giocattoli. Solo con la rivoluzione industriale e lo sviluppo della meccanica e della chimica, vennero costruiti ed impiegati i primi automi meccanici in ambiti lavorativi al fianco degli esseri umani.

Intorno al secolo XX il termine automa viene sostituito con il termine robot, così, alcuni elettrodomestici come il macina caffè o il frullatore prendono la denominazione di "robot da cucina". Caratteristica di tali robot, però, é quella non solo di avere un set di operazioni limitate e pre-fissate, ma anche di avere sempre bisogno della presenza dell'uomo per essere usati o pre-impostati. La visione moderna di robot prevede esseri artificiali con sembianze sempre più simili a quelle dell'uomo, spesso anche dotati di articolazioni che ne aumentano i movimenti e le funzionalità. Inoltre l'intelligenza artificiale di cui sono forniti li rende sempre più capaci di "pensieri autonomi" smettendo così di eseguire algoritmi, che pur essendo complessi, rimangono frutto di una elaborazione dell'uomo. Tale genere di robot é munito di molteplici sensori ed attuatori, con i quali poter interagire con l'ambiente circostante, e anche collaborare attivamente con l'uomo nella vita di tutti i giorni.

0.2 Obiettivi

Obiettivo di questo elaborato é proporre di un sistema di controllo per robot mobili basato su ReSpect Situated. Tale sistema dovrà rendere il robot autonomo e capace di muoversi in maniera intelligente

in qualsiasi ambiente egli si trovi. Si studierà in oltre, come caso specifico di applicazione, la tecnologia Lego Mindstorm NXT.

Capitolo 1

Intelligenza artificiale e robotica

Gli sviluppi che interessano la nascita dell'intelligenza artificiale avvengono attorno alla metà del Novecento per opera di Alan Turing ¹ che oltre al modello ideale di calcolatore automatico "universale" (la *macchina di Turing*), propose il "*gioco dell'imitazione*", ossia un paradigma per stabilire se una macchina é "intelligente". Nel suo noto articolo *Computing Machinery and Intelligence* (1950), egli suggeriva di porre un osservatore di fronte a due telescriventi. Una delle due é comandata da un uomo, l'altra da un calcolatore programmato in modo da "fingere" di essere una persona umana. Quando non si riuscirá a distinguere il calcolatore dell'interlocutore umano, allora si potrà dire che il calcolatore é "intelligente".

Il termine Intelligenza Artificiale (IA) venne proposto nel 1956 da John McCarthy ², in occasione di uno storico incontro organizzato presso il Dartmouth College ³. L'Intelligenza Artificiale si propone di indagare i meccanismi che sono alla base della cognizione umana, in primo luogo il ragionamento logico-matematico, la capacità di risolvere problemi e la comprensione del linguaggio naturale, con il fine dichiarato di riprodurli per mezzo di elaboratori elettronici sufficientemente potenti.

La posizione di partenza della IA é costituita dal fatto che molti scienziati considerano il pensiero umano come una forma di calcolo e,

¹nato a Londra, 23 giugno 1912 – morto a Wilmslow, 7 giugno 1954. É considerato uno dei padri dell'informatica.

²scienziato informatico. Ha vinto il Premio Turing nel 1971 per i suoi contributi nel campo dell'A.I. .

³colleg in Hanover, New Hampshire

in quanto tale, potenzialmente riproducibile tramite computer dotati di adeguati software. Da questo punto di vista si possono distinguere due posizioni principali: quella dell'intelligenza artificiale **forte** e quella dell'intelligenza artificiale **debole**.

I sostenitori della tesi **forte** dell'intelligenza artificiale sono convinti che il pensiero sia interamente riconducibile a un processo di manipolazione di simboli che si attua utilizzando un gran numero di algoritmi ⁴. Pensare, in tale prospettiva equivale a calcolare, da cui deriva che la mente é equivalente al programma di un calcolatore.

Per i sostenitori della versione debole dell'intelligenza artificiale, pensare non é sinonimo di calcolare, poiché le capacità del pensiero non si limitano a quelle logico-matematiche. Il calcolatore non é visto come l'analogo di un cervello biologico, ma come uno strumento che può essere di grande aiuto nella comprensione dei processi cognitivi.

Uno dei principali campi di applicazione dell'intelligenza artificiale é certamente il campo della **robotica**.

La parola robotica proviene dal ceco *robota*, dove ha il significato di "lavoro pesante" o "lavoro forzato". La robotica é una scienza che studia i comportamenti degli esseri intelligenti, cerca di sviluppare delle metodologie che permettano ad una macchina (robot), dotata di opportuni dispositivi atti a percepire l'ambiente circostante ed interagire con esso quali sensori e attuatori, di eseguire dei compiti specifici. É una disciplina relativamente nuova, che affonda le sue radici nell'antico desiderio dell'uomo di costruire strumenti che possano liberarlo da compiti troppo faticosi, noiosi o pericolosi. La scienza robotica trova applicazioni in molteplici contesti, questo ha fatto sì che nascessero varie sotto-discipline fra le quali però raramente esiste una netta linea di demarcazione.

1.1 Le leggi della robotica

Le leggi, inizialmente tre, sono poi state completate in quattro con l'aggiunta di una legge 0. Dette leggi codificano le norme etico/comportamentali che un qualsiasi robot deve rispettare. L'organizzazione delle leggi, piuttosto che identificare compiti specifici che un robot deve assolvere, si limita a definire il ruolo che questi strumenti possono assumere all'interno della società, vincolandone la propria autonomia di com-

⁴procedimento che consente di ottenere un risultato atteso eseguendo, in un determinato ordine, un insieme di passi semplici corrispondenti ad azioni scelte solitamente da un insieme finito.

portamento, rispettivamente alla sicurezza dell'uomo e dell'umanità, quindi alla sua manifestazione di volontà:

- **Legge 0:** Un robot non deve provocare danno all'umanità sia tramite la sua azione che tramite un comportamento passivo *questa legge è stata aggiunta in seguito.*
- **Legge 1:** Un robot non deve ferire esseri umani o tramite la sua non azione consentire un danno agli stessi.
- **Legge 2:** Un robot deve obbedire agli ordini degli esseri umani eccetto quando questi siano in contrasto con la prima legge.
- **Legge 3:** Un robot deve agire per proteggere la sua stessa esistenza fintanto che tali azioni non siano in contrasto con le prime due leggi.

1.2 Automi e robot

La possibilità di creare degli **automi**, ovvero macchine che effettuino con efficienza lavori prestabiliti, è evidente, ciò che è un miraggio lontano è la possibilità di infondere coscienza e senso comune in tali macchine.

Il termine **robot**, che nel 1920 sostituì il termine automa, deriva dal ceco "robota", che significa "lavoro pesante" o "lavoro forzato". Nel linguaggio comune, un robot è un'apparecchiatura artificiale che compie determinate azioni in base ai comandi che gli vengono dati e alle sue funzioni, sia sotto la supervisione diretta dell'uomo, sia autonomamente basandosi su linee guida generali, magari usando processi di intelligenza artificiale. Questi compiti tipicamente dovrebbero essere performati al fine di sostituire o coadiuvare l'uomo, come ad esempio nella fabbricazione, costruzione, manipolazione di materiali pesanti e pericolosi, o in ambienti proibitivi o non compatibili con la condizione umana, o semplicemente per liberare l'uomo da impegni.

Un robot così definito, dovrebbe essere dotato di connessioni guidate dalla retroazione tra percezione e azione, e non dal controllo umano diretto. L'azione può prendere la forma di motori elettro-magnetici, o attuatori, che muovono un arto, aprono e chiudono una pinza, o fanno deambulare il robot. Il controllo passo-passo e la retroazione sono forniti da un programma che viene eseguito da un computer esterno

o interno al robot, o da un microcontroller. In base a questa definizione, il concetto di robot può comprendere quasi tutti gli apparati automatizzati.

In alternativa, il termine robot viene usato per indicare un essere artificiale, un automa o androide⁵, che replichi e somigli ad un animale ad un uomo. Il termine viene utilizzato per definire molte macchine che sostituiscono direttamente un umano o un animale, nel lavoro o nel gioco. In questo modo, un robot può essere visto come un tentativo di biomimica.

Ad oggi potremo definire ‘Robot’ un sistema elettromeccanico riprogrammabile, dotato di capacità di percezione e di un’intelligenza propria, predisposto per compiere un ampio numero di compiti diversi. I Robot si differenziano per applicazione, struttura ed attuazione in diverse categorie, alcuni esempi sono:

- Biorobotica: robot usati in campo medico.
- Robotica militare
- Robotica marina
- Robotica biomedicale: ramo della robotica molto vasto comprende diversi tipi di robot; si va infatti dai robot capaci di assistere il medico durante le operazioni chirurgiche a quelli che tramite tecnologie di
- Robotica spaziale: esempi di questi robot sono le sonde esplorative impiegate in diverse missioni sui pianeti del sistema solare ma anche robot più tradizionali come il famoso braccio manipolatore dello Space Shuttle.

1.3 Uso contemporaneo dei robot

I robot utilizzati adesso sono in realtà dei computer muniti di servomeccanismi; esistono moltissime tipologie di robot differenti sviluppate per assolvere i compiti più disparati. Ormai è larghissimo l’impiego dei robot nell’industria metalmeccanica e non solo. Si possono catalogare i robot in due macro categorie: **autonomi** e **non autonomi**.

I robot **non autonomi** sono i classici robot utilizzati per adempiere a specifici compiti che riescono ad assolvere in maniera più efficace

⁵essere artificiale con sembianze umane

dell'uomo; alcuni casi sono i robot utilizzati nelle fabbriche con l'enorme vantaggio di poter ottenere una produzione più precisa, veloce ed a costi ridotti; oppure i robot utilizzati per lavorare in ambienti ostili (ad esempio su Marte) o con sostanza tossiche; questi robot sono detti "non autonomi" poiché sono guidati da un software deterministico che fa eseguire loro il lavoro in modo ripetitivo, oppure sono direttamente pilotati dall'uomo.

I robot **autonomi** sono invece caratterizzati dal fatto che operano in totale autonomia ed indipendenza dall'intervento umano e sono in grado di prendere decisioni anche a fronte di eventi inaspettati. Questi Robot sono programmati solitamente con algoritmi che si rifanno a tecniche di intelligenza artificiale: algoritmi genetici, logica fuzzy, learning, e come ultima frontiera anche reti neurali. I robot autonomi sono adatti a svolgere compiti in ambienti non noti a priori; tipicamente si tratta di robot mobili. Alcuni piccoli robot autonomi vengono comunemente utilizzati per il taglio dell'erba nei giardini: essi autonomamente decidono quando partire, dove tagliare e quando tornare alla base per ricaricarsi.

1.3.1 La robotica umanoide

Sin dall'inizio della coniazione del vocabolo, il termine robot è stato strettamente associato a caratteristiche di sistemi elettromeccanici di tipo antropomorfi. La stessa rappresentazione del R.U.R.⁶ raffigurava un manichino dalla forma umana. Durante gli ultimi 50 anni, narrativa e cinematografia hanno prevalentemente immaginato la figura del robot come un essere meccanico dalle sembianze umane. L'idea di ricostruire un robot dalle forme umane, è oggi solo una delle linee di ricerca della robotica moderna, la robotica umanoide, ma ciononostante costituisce tuttoggi una delle sfide più ambiziose sia per le complessità di percezione e meccaniche insite, sia per la difficoltà di ricreare il coordinamento motorio che per l'intelligenza propria dell'azione di esseri umani. I primi esperimenti di progettazione di robot dalle forme umane risalgono alle ricerche della Waseda university dei primi anni '70. Nonostante un trentennio abbondante di ricerca nel settore, tuttavia la realizzazione di un sistema che si avvicini alle prestazioni umane è ancora lontana. I due risultati più di successo nel settore sono il robot umanoide realizzato dalla Honda e presentato nel 1998 e il Sony Dreaming Robot, oggi alla sua 4 versione. Entrambi hanno capacità di locomozione e equilibrio, e sono in grado di

⁶Rossum's Universal Robots, opera scritta dal drammaturgo ceco Karel Capek.

eseguire dei compiti elementari in accordo a traiettorie e movimenti preimpostati.



Figura 1.1: *Asimo, robot umanoide realizzato dalla Honda.*

1.3.2 Reti neurali

Le reti neurali sono ispirate al sistema nervoso, nel senso che una rete é un insieme di nodi, di unità, che sono simili, in un modello semplificato, alle cellule nervose. Queste unità sono collegate fra loro da connessioni che simulano i collegamenti sinaptici fra neuroni. Attraverso tali connessioni questi neuroni artificiali, emettendo eccitazioni o inibizioni, rappresentano il modo in cui ogni neurone comunica con gli altri. Tale modalità é dovuta però al peso particolare di ciascuna connessione. L'apprendimento, ad opera di una rete neurale, equivale, in sostanza, al modificarsi dei pesi sulle connessioni di modo da produrre capacità che la rete non presentava all'inizio dell'apprendimento.

Mentre l'intelligenza artificiale tradizionale cerca di riprodurre le capacità degli esseri umani in termini di simboli e regole per manipolare questi simboli, le reti neurali sono invece modelli ispirati alle caratteristiche fisiche del corpo, in particolare del sistema nervoso. Una differenza cruciale, per esempio, consiste nel fatto che un sistema di intelligenza artificiale in genere non apprende, perché é previamente programmato. Al contrario, é tipico delle reti neurali e, in genere, di tutti i sistemi che si ispirano alle caratteristiche degli organismi biologici, che le capacità di tali sistemi sono frutto di un apprendimento autonomo, spontaneo, avvenuto all'interno del sistema. C'è, quindi, una forma di auto organizzazione, di auto apprendimento, che non esiste nei sistemi tradizionali . Queste forme di autoapprendimento,

fanno sì che lo stesso sistema trovi da solo le soluzioni più interessanti a determinati problemi a cui noi non avremmo pensato se avessimo dovuto programmarlo.

1.4 Linee di sviluppo della robotica

La robotica prende piede nella società negli anni '70 come supporto alla produzione industriale. In quel periodo, l'ambiente in cui opera il robot e l'ambiente in cui opera l'uomo sono completamente separati al fine di garantire i margini di sicurezza agli operatori necessari. I primi robot inoltre operavano in ambienti completamente strutturati, ovvero dove le posizioni di tutti gli elementi con cui il robot doveva interagire erano note a priori. Successivi sviluppi della robotica, come la visione artificiale, hanno poi consentito di ridurre i vincoli imposti nell'ambiente. Successivamente la robustezza e i fattori di sicurezza per gli operatori collegati all'impiego dei robot hanno indotto i ricercatori a progettare nuovi sistemi, detti di teleoperazione, in cui i robot siano capaci di trasportare in ambienti remoti e/o ostili le capacità di azione e destrezza di un operatore umano. L'avvento della teleoperazione ha introdotto nel paradigma di azione dei robot una profonda trasformazione dal momento in cui l'operatore doveva operare nello stesso spazio fisico del robot con cui interagisce. Negli anni '90, gli ambienti virtuali hanno ulteriormente ridotto questa distanza tra operatore e robot, immaginando e realizzando una serie di dispositivi robotici (dette 'interfacce afferenti') tramite i quali l'operatore risulta in grado di interagire con l'ambiente virtuale e percepirne stimoli fisici. L'uso di questi dispositivi (aptici, tattili, termici) richiede che l'operatore indossi fisicamente una componente o tutto il meccanismo, in grado di trasmettere percezioni tramite aree di contatto. Gli ambienti virtuali arricchiscono quindi la capacità di condividere l'ambiente tra sistema robotico e uomo, con la capacità di condividere le esperienze. Un ulteriore sviluppo di questi sistemi viene dall'intelligenza artificiale, ad oggi infatti, un'ampia serie di sistemi robotici, consentono di condividere con l'operatore non solo l'interazione in termini di esperienza, ma anche in termini di intenzione. È possibile suddividere le aree di ricerca ed applicazione della robotica in quattro grandi categorie:

- **robot industriali**, i primi in ordine di applicazione, hanno oggi un vastissimo impiego in tutti i settori della produzione. Essi infatti consentono di ridurre costi e tempi di produzione e di poter modulare i costi in base ai volumi di produzione necessari.

In aggiunta miglioramenti qualitativi dei prodotti, e la maggior sicurezza per il personale danno forti motivazioni a questo tipo di produzione. Nel campo della robotica industriale esistono manipolatori per decine di applicazioni differenti, solo per citarne alcune: verniciatura, assemblaggio, pulizia, ispezione, saldatura, montaggio, movimentazione, taglio, controllo qualità, rilevamento,... ed il numero di applicazioni sembra destinato ad aumentare con il tempo e l'autonomia raggiunta da questi dispositivi.

- **robot sociali**, individuano una nuova applicazione della robotica destinata ad essere nel futuro uno strumento di interazione sociale. Eliminata la barriera della sicurezza, tramite una serie accurata di norme e di certificazioni, i robot possono entrare a far parte, come già i computer hanno fatto 20 anni fa, della vita sociale. Rispetto ai computer questi ultimi sembrano avere maggiori potenzialità espressive nel fatto che possono integrare le capacità multimodali con movimento e gestualità ai primi negati. Ad oggi lo sviluppo dei robot sociali è ancora limitato, si va da alcuni esempi commerciali a risultati perlopiù sperimentali.
- **Robot per il gioco** (il Sony Aibò, Mitsubishi Wakamaru), per **il cinema** (l'animatronica) e **l'assistenza agli anziani** (domotica), sono ormai diffusi.
- Di più recente ricerca invece sono i robot per **l'interazione artistica, lo sport e la musica**.

In ambito medico, l'impiego della robotica sta rivoluzionando sia la pratica chirurgica (Computer Aided Surgery) che quella di clinica (fisioterapia, assistenza tecnologica). Nel corso degli ultimi dieci anni, diversi sistemi robotici sono stati progettati e commercializzati per completare gli apparati di chirurgia mini-invasiva, con strumentazione robotica in grado di rendere naturale ed intuitivo il processo di intervento. In ambito terapeutico esistono ad oggi diversi sistemi di riabilitazione neuro-motoria basati su interfacce robotiche controllate al calcolatore. Questi sistemi possono essere applicati in diversi ambiti a seconda della patologia clinica. Ad esempio, smorzatori attivi del tremore possono essere impiegati in caso di sclerosi multipla o Parkinson; sistemi di registrazione ed ausilio motorio possono essere impiegati per esercizi post-traumatici o per particolari patologie che danneggiano le capacità motorie.

1.5 Componenti di un robot

La realizzazione ed il funzionamento vengono ottenuti tramite l'integrazione di una serie di componenti diverse così definite: parti meccaniche, sistema di trasmissione, sistema di attuazione, elettronica di controllo e rilevazione, sistema di calcolo e controllo, software comportamentale.

- Le componenti meccaniche individuano le parti strutturali della macchina. Le parti strutturali vengono a loro volta suddivise in due categorie: i *link* ed i *giunti*. I primi costituiscono il corpo del robot, i secondi le articolazioni mobili. Due sono i tipi di giunti base, rotativo e lineare. Alla progettazione della meccanica di un robot contribuiscono specifiche prestazionali ed operative determinate dalla specifica applicazione cui esso è destinato. A seconda della combinazione di giunti e di link, si potrà parlare di diverse categorie quali: robot seriale, seriale diramato, parallelo, ibrido.
- La trasmissione si occupa di trasmettere l'energia prodotta da un sistema di attuazione alle articolazioni del robot. Esistono diverse tecnologie tramite le quali tale trasmissione può essere determinata: diretta (il motore è sull'asse del giunto), a cinghia, a cavi ecc. ecc.
- L'attuazione invece è la sorgente di energia che fornisce movimento al robot. Gli attuatori si distinguono in due classi: lineari e rotativi. I primi possono agire in senso alternato lungo una linea di movimento, i secondi ruotare attorno un asse. Inoltre l'attuazione si distingue sia per classificazione fisica del mezzo di trasmissione dell'energia (elettrica, idraulica, pneumatica) che per specifico principio di trasduzione.
- L'elettronica di un robot è forse la componente con maggiori varianti possibili. Essa si distingue in due parti: la componente di acquisizione e quella di attuazione. La prima serve a raccogliere informazioni ambientali (quali la posizione del robot, forze di contatto, accelerazioni, visione, temperatura,...) la seconda a trasformare le indicazioni del sistema di controllo in segnali elettrici idonei a pilotare i motori. I sistemi di attuazione (driver) sfruttano una relazione diretta tra una variabile elettrica controllabile e l'azione meccanica da esercitare. Nel caso dei motori elettrici generalmente si controlla la relazione tra

la tensione-corrente presentata al motore e la coppia/velocità di esercizio.

- Il controllo di un robot si occupa di coordinare i segnali elettronici rilevati dal sistema di percezione per produrre segnali di movimento da trasdurre in opportuni comandi elettrici per i motori. Sebbene originariamente siano stati creati anche meccanismi di controllo meccanici, due categorie di controlli sono attualmente in uso: controlli elettronici e controlli digitali. La tendenza, sia per questioni di costo che di affidabilità è comunque nel digitalizzare tutti i controlli.
- Il software comportamentale definisce infine il compito e le relazioni del robot con l'ambiente circostante. Diverse famiglie di software comportamentali sono disponibili e si differenziano prevalentemente in base al campo di applicazione. Alcuni esempi includono: inseguimento di traiettorie preimpostate (automazione industriale) ovvero fornite in tempo reale (teleguida, teleoperazione), rilevazione di parametri (controllo qualità, ispezione), analisi semantica dei dati rilevati (sorveglianza, navigazione autonoma), inseguimento di fattori di merito (ausili tecnologici, extender,...). Inoltre il software comportamentale include un'ampia famiglia di soluzioni analitiche/numeriche per risolvere specifici problemi di movimentazione: cinematica diretta, cinematica inversa, pianificazione di traiettorie.

1.6 Alcuni esempi

Il Museo Delle Forme Pure (www.pureform.org) è un sistema in cui un dispositivo robotico viene integrato con un ambiente virtuale. Un Museo, composto di sale e statue, esatte copie digitali dei relativi originali, viene materializzato alla vista del visitatore e alla percezione delle sue mani. Una vera e propria esperienza artistica virtuale che, grazie alla possibilità di toccare le opere, offerta dalla tecnologia aptica, si apre a nuove esperienze e nuove utenze come quella dei non vedenti.

La scrivania aptica è un sistema per integrare le classiche interfacce grafiche al calcolatore con strumenti di interazione robotica. L'uso della robotica e della percezione fisica in questo caso può essere veicolo di trasmissione di informazione utile per la messa a punto di

software formativi in grado di insegnare particolari abilità motorie al suo utilizzatore (scrivere, disegnare, etc.).

Il Body Extender è un particolare dispositivo robotico antropomorfo in grado di essere indossato dal suo manovratore. Il sistema robotico prevede strutture in grado di assistere il movimento delle gambe e delle braccia dell'operatore nella manovra di grossi carichi (in termini di peso) e di 'alleggerirli' fisicamente tramite un sistema di controllo in tempo reale.



Figura 1.2: *Utente non vedente che, tramite l'ausilio di un robot, riesce a godere delle opere d'arte esposte al Museo Delle Forme Pure.*



Figura 1.3: *Alcuni esempi di Body Extender.*

Capitolo 2

Sistema di controllo per un robot mobile

Un sistema di controllo per un robot mobile deve essere in grado di processare numerose operazioni in real-time. Tali operazioni devono essere eseguite in un ambiente dinamico, i cui contorni non solo cambiano rapidamente, ma spesso sono difficili da delineare a causa dei disturbi che raggiungono i sensori del robot adibiti alla mappatura dell'ambiente. L'approccio usuale per costruire tali sistemi di controllo è decomporre il problema in una serie di *unità funzionali* orizzontali :

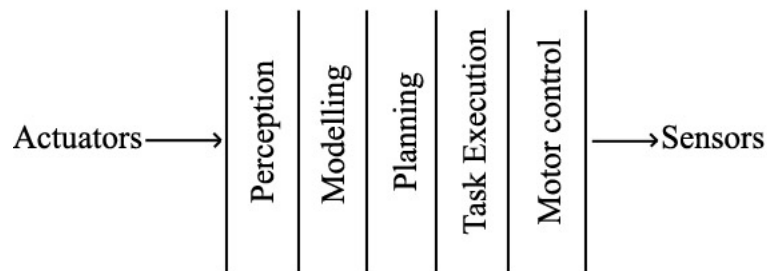


Figura 2.1: *Decomposizione tradizionale di un sistema di controllo in moduli funzionali*

Alternativamente può essere effettuata una suddivisione verticale, che esprime non tanto le funzionalità che il sistema di controllo

deve avere al suo interno, bensí i comportamenti che il robot deve manifestare durante l'interazioni con il mondo esterno:

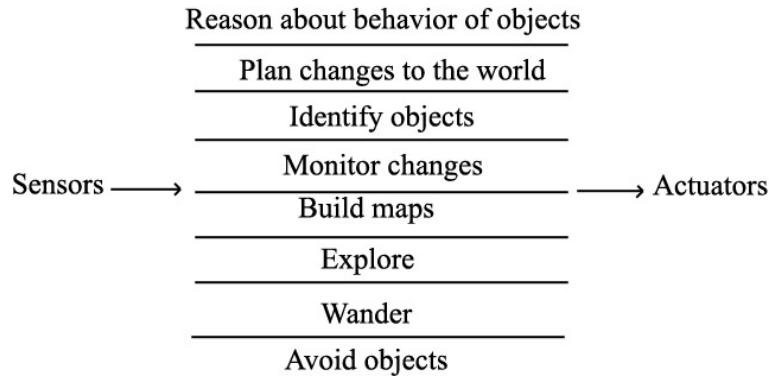


Figura 2.2: *Decomposizione di un sistema di controllo basata sui comportamenti che un robot deve manifestare.*

2.1 Requisiti

Possono essere identificati alcuni requisiti base che un sistema di controllo deve soddisfare:

- *Obiettivi multipli.* Spesso i robot hanno piú compiti da soddisfare contemporaneamente. Ad esempio, un robot deve avere la capacità di evitare ostacoli mentre sta seguendo un itinerario fra un punto di partenza e uno di arrivo; potrebbe cercare di raggiungere un punto nel tempo minimo preservando piú carica possibile, e cosí via. Spesso l'importanza che un compito ha sugli altri é relativa al contesto in cui ci si trova, per questo un buon sistema di controllo deve essere in grado di gestire in maniera intelligente anche le priorità degli obiettivi da portare a termine.
- *Sensori multipli.* I robot moderni sono tutti dotati di piú sensori, ognuno dei quali specializzato al rilevamento di una determinata

grandezza fisica proveniente dall'ambiente esterno. Tutti i sensori, anche i piú moderni, hanno una componente di errore nelle proprie rilevazioni, possono verificarsi infatti casi di misurazioni sballate, ma anche situazioni di inconsistenza fra una misurazione e l'altra a causa di una poca adeguata scelta delle unità di misura. Nonostante ciò, un robot deve riuscire a prendere ugualmente delle decisioni.

- *Robustezza.* Un robot deve essere un entità robusta. Quando un sensore si rompe, esso deve essere in grado di continuare ad eseguire i propri compiti affidandosi a quelli rimasti. Ancora, quando l'ambiente esterno cambia in maniera drastica il robot dovrebbe continuare ad eseguire una serie di operazioni base, piuttosto che vagare senza scopo o rimanere immobile. Idealmente un robot dovrebbe continuare a garantire le funzioni base anche in presenza di errori e/o guasti.
- *Additività.* Il numero di sensori di un robot deve poter variare a seconda delle esigenze, elemento importante però é che l'aggiunta di nuovi sensori non detragga energia agli altri, impedendone un buon funzionamento. Un numero di sensori variabili comporta la necessità di variare anche l'energia erogata dal robot.

2.2 Livelli e strati

Ci sono molti approcci che consentono la realizzazione di un robot intelligente ed autonomo. Come la maggior parte dei problemi ingegneristici, partiamo con lo scomporre il problema in parti, ed analizzare le varie soluzioni per ognuno di esse.

2.2.1 Livelli di competenza

La suddivisione tipica che viene fatta dai progettisti di sistemi di controllo per robot é una suddivisione orizzontale, le informazioni attraversano tutto il sistema di controllo, partendo dalle percezioni dell'ambiente, sino ad arrivare ai motori, per poi creare un feedback di ritorno tramite il quale il robot agisce sull'ambiente stesse. I principali livelli di questa suddivisione sono:

- *Percezione.*

- *Mappatura dei dati.* (Tramite i quali viene creata una rappresentazione del mondo esterno)
- *Pianificazione.*
- *Esecuzione dei compiti.*
- *Controllo dei motori.*

La scelta che é stata fatta per realizzare il nostro sistema di controllo, invece, é quella di adottare una suddivisione verticale, e, anziché concentrarsi sulle principali funzionalità interne, sono state analizzate le principali manifestazioni esterne che il sistema di controllo deve avere. Per fare questo abbiamo definito un certo numero di "livelli di competenza" per un robot mobile autonomo. Ogni livello di competenza é una rappresentazione informale di una desiderata classe di comportamento che vogliamo venga manifestato dal robot a prescindere dall'ambiente in cui esso si trova. Un piú alto livello di competenza implica una piú alta specializzazione del comportamento. Sono stati utilizzati i seguenti livelli di competenza come guida per il progetto del sistema di controllo:

1. Evitare collisioni con oggetti, sia mobili che in movimento.
2. Vagare nell'ambiente, sempre prestando attenzione a non urtare eventuali ostacoli.
3. Esplorare ed osservare il mondo, valutando le distanze dai luoghi raggiungibili ed, eventualmente, dirigersi verso di essi.
4. Costruire una mappatura dell'ambiente tramite i dati letti dai sensori, e pianificare dei tragitti da un punto ad un altro.
5. Rilevare cambiamenti nell'ambiente circostante.
6. Riconoscere gli oggetti dell'ambiente, ed eseguire operazioni specifiche su di essi.
7. Pianificare ed eseguire operazioni che portano l'ambiente a modificazioni desiderate.
8. Ragionare sul comportamento degli oggetti dell'ambiente, e reagire di conseguenza, modificando, eventualmente, il proprio comportamento.

2.2.2 Strati di controllo

L'idea chiave dei livelli di competenza é che possiamo costruire strati di un sistema di controllo corrispondenti ad ognuno di essi, e, semplicemente, aggiungere nuovi strati a quelli già esistenti per raggiungere livelli di competenza maggiori. Partiremo con il creare un sistema di controllo composto da un solo strato, il quale quindi implementerà il primo livello di competenza.

Tale livello sarà chiamato Livello 0, grazie al quale il robot sarà in grado di evitare gli ostacoli. Per tale motivo il Livello 0 sarà sempre in esecuzione, dovrà quindi essere ben testato e debuggato per garantire una buona affidabilità e robustezza. Una volta realizzato tale livello, andremo ad aggiungere il livello di competenza successivo, ovvero il così detto Livello 1. Esso si appoggerà al Livello 0, sarà in grado di analizzarne i dati e di inserirne di nuovi, sopprimendo il normale flusso informativo del livello sottostante. Questo meccanismo può essere eseguito per i successivi livelli di competenza che si vorranno aggiungere al sistema di controllo base.

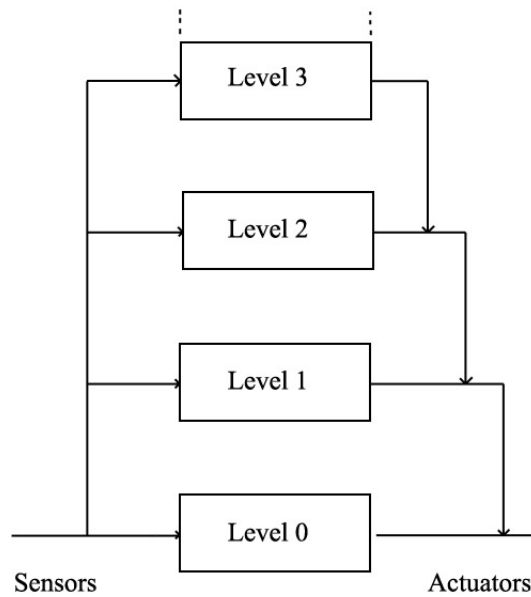


Figura 2.3: *Sistema di controllo a strati verticali. Il sistema é funzionante già con il completamento del Livello 0, tutti gli altri sono opzionali.*

Questa architettura prende il nome di "architettura ad inclusione". Essa soddisfa numerosi requisiti per un sistema di controllo sopra

citati, infatti:

- *Obiettivi multipli.* Ogni livello di competenze é indipendente l'uno dell'altro, e posso quindi perseguire obiettivi diversi nello stesso momento. Grazie al meccanismo della soppressione del flusso dati, fatto dai livelli superiori a discapito di quelli inferiori, é inoltre possibile garantire che i compiti del robot vengano svolti con prioritá adeguate.
- *Sensori multipli.* Il problema di eventuali errori di lettura dei sensori ovviamente prescinde dal tipo di architettura utilizzata. Si pone però rimedio ai problemi di inconsistenza dei dati letti, infatti, allo stesso tempo il robot può effettuare diverse rilevazioni, e il suo sistema di controllo non deve preoccuparsi di elaborarle tutte, ma semplicemente deve recapitare le informazioni giuste ai livelli di competenza giusti, i quali avranno poi il compito di elaborare i dati in maniera adeguata, e di passarli agli strati inferiori.
- *Robustezza.* Non solo un numero elevato di sensori produce un grado di robustezza maggiore. Nell'architettura proposta, i livelli piú bassi, che sono stati debuggati e testati molto bene, continuano a rimanere in esecuzione quando quelli piú alti vengo aggiunti. I livelli superiori possono modificare il normale flusso di dati dei livelli inferiori, sopprimendone ad esempio l'output, ma nel caso in cui tale operazione non vada a buon fine, il livello piú basso continuerá comunque a produrre dati attendibili garantendo le funzionalitá base del robot.
- *Additività.* Questo problema é in parte risolto dal fatto che ogni livello é un mondo a parte, non dipende da altri. l'ideale sarebbe poter fornire, ad esempio, ogni livello di un processore di calcolo dedicato.

2.3 Struttura degli strati

Ma come andremo ad implementare ogni singolo strato del nostro sistema di controllo? Abbiamo veramente la necessitá di decomporli nella maniera tradizionale? La risposta é no. Siamo liberi di usare differenti scomposizioni per differenti set di compiti che ogni singolo strato dovrá implementare.

Abbiamo scelto di costruire ogni strato partendo da un piccolo set di moduli, i quali si scambiano messaggi l'un l'altro. Non é previsto alcun messaggio di acknowledgement in tali comunicazioni, é lasciata, quindi, la possibilitá che i dati siano persi (cosa che puó avvenire anche frequentemente), questo per non compromettere la reattività del robot.

Ogni modulo é una macchina a stati finiti, che implementa una parte del comportamento del livello di competenza in questione. Essi lavorano in maniera completamente asincrona, monitorando i propri canali di ingresso, e scrivendo in quelli di uscita. Non esistono altre forme di comunicazione, in particolare non vi é la presenza di alcuna memoria condivisa. Ogni modulo viene creato nello stesso modo, nel senso che non esiste una centralizzazione all'interno dello strato di controllo, semplicemente ogni modulo cerca di eseguire i propri compiti nel miglior modo possibile. Gli input di un modulo possono essere inibiti, e gli output soppressi da segnali provenienti da altri moduli, questo é il meccanismo tramite il quale i livelli piú alti riescono ad "inglobare" il comportamento di quelli piú bassi.

2.3.1 Comunicazione fra moduli

La figura 3.4 mostra in maniera migliore come ogni modulo puó comunicare con gli altri. Essi hanno diverse linee di ingresso ed altrettante linee di uscita. Una linea di output di un modulo é connessa ad una o piú linee di input di un altro modulo. Tali linee possono essere visti come dei fili, con una sorgente ed una destinazione. In aggiunta gli output possono essere inibiti, e gli input soppressi.

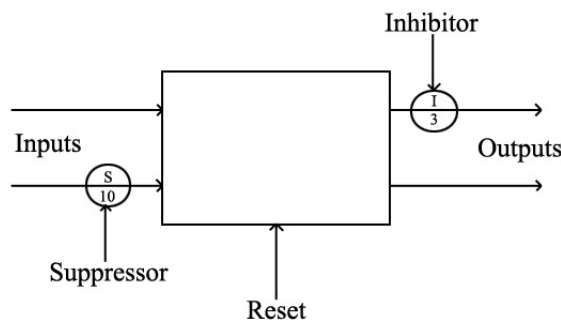


Figura 2.4: *Meccanismo di comunicazione di un modulo.*

L'output di un modulo può terminare sulla linea di output di un altro modulo; ogni segnale di output di quest'ultimo verrà inibito dal primo. In maniera analoga, il segnale di output di un modulo può terminare sulla linea di input di un altro modulo, in questo caso gli input vengono soppressi, ma a differenza di prima essi vengono rimpiazzati dai nuovi input. Per entrambi le operazioni, di inibizione e soppressione, vengono specificati i tempi di durata.

Vediamo per esempio, in pseudo codifica, come il modulo Avoid (del Livello 1 di competenze) sopprime l'input del modulo Motor (del Livello 0 di competenze) per 1,5 secondi:

```
(defwire (Avoid command), ((suppress (Motor command) 1,5 )))
```

2.4 Livello 0

Il livello più basso del sistema di controllo si preoccupa di non fare urtare il robot con oggetti che lo circondano. Questo definisce quindi il primo livello di competenza. Se qualcosa cerca di avvicinarsi al robot, esso scappa, se durante il suo percorso si avvicina ad un ostacolo, si ferma. Questi sono i comportamenti base che vogliamo vengano manifestati dal robot. Ovviamente il nostro robot non è invincibile, perciò un oggetto che si avvicina troppo rapidamente potrebbe colpirlo, come un ambiente troppo pieno di ostacoli potrebbe non essere percorso senza urti. Ma andiamo ad analizzare i moduli che compongono il nostro primo strato di controllo:

- Il modulo *Motor* comunica direttamente con gli attuatori. Esso deve essere fornito di un meccanismo di comunicazione particolare che gli consenta di ricevere ed inviare comandi direttamente al robot fisico. Egli accetta comandi contenenti la velocità alla quale i motori devono girare, e l'angolazione di rotazione delle ruote. Se qualche messaggio di *halt* viene ricevuto, esso ordina ai motori di fermarsi immediatamente (tutte le eventuali altre informazioni inviatogli in questo caso vengono perse).
- il modulo *Sonar* si preoccupa di ricevere le informazioni direttamente del sensore ad ultrasuoni contenenti la distanza degli ostacoli. Grazie a questa informazione il nostro sistema di controllo può effettivamente crearsi una sorta di mappatura dell'ambiente circostante.

- il modulo *Collide* monitora le informazioni passategli dal modulo sonar, e se individua un oggetto in prossimità del robot manda un segnale di halt al modulo motor. *Collide* non sa e non si preoccupa se il robot si muove o è immobile, i messaggi di halt che vengono generati mentre il robot non si muove vengono essenzialmente perduti.
- il modulo *FeelForce*. dopo aver ricevuto informazioni dal sonar, elabora le distanze dagli oggetti, e produce informazioni relative alla velocità e l'angolazione che il robot deve avere per continuare a muoversi evitando eventuali ostacoli.
- il modulo *RunAway* riceve le direttive per il robot dal modulo *FeelForce*, le elabora, e se le modifiche da apportare alla velocità e all'angolazione di rotazione del robot sono significative, vengono passate al modulo motor che si incaricherà di avvertire i motori fisici.

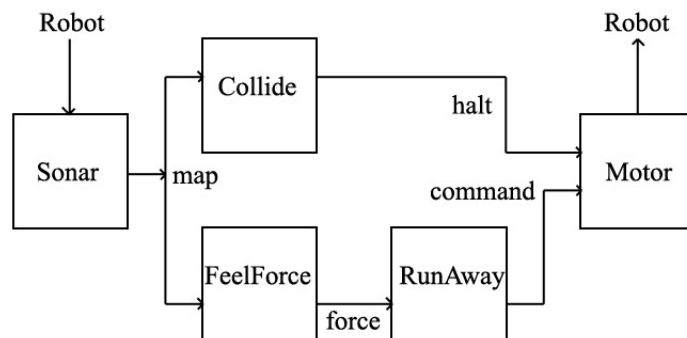


Figura 2.5: *Struttura dell Livello 0* .

2.5 Livello 1

Il Livello 1 del nostro sistema di controllo, quando combinato con il Livello 0, conferisce al robot la capacità di muoversi liberamente nell'ambiente senza urtare oggetti. I moduli che lo compongono sono solo due :

- Il modulo *Wander* il quale genera una direzione e una velocità casuale ogni 10 secondi circa.

- il modulo *avoid* che prende le informazioni computate dal Livello 0 dal modulo *FeelForce*, le combina con i comandi generati dal modulo *Wander*, e genera un ulteriore comando, contenente velocità e rotazione del robot che rappresenta una sorta di mix ottimale fra la direzione selezionata da *Wander*, e quella generata da *FeelForce*. In questo modo il robot può spostarsi nell'ambiente, seguendo una sorta di itinerario casuale, sempre però continuando ad evitare gli ostacoli. da notare che la linea di output del modulo *Avoid* va a sopprimere l'ingresso di *Motor* il quale non riceverà più informazioni da *RunAway* per un tempo prestabilito.

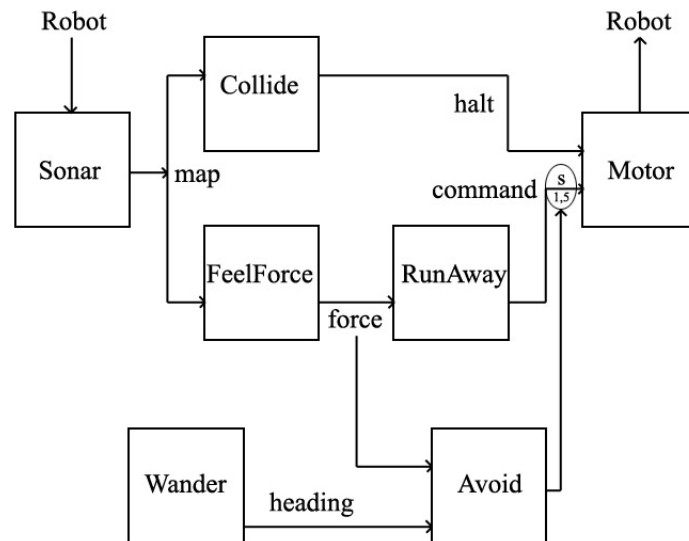


Figura 2.6: *Struttura dell Livello 0 aumentato anche del Livello 1 .*

Capitolo 3

Lego Mindstorms

Lego Mindstorms é una linea di prodotti LEGO che combina mattoncini programmabili con motori elettrici, sensori, mattoncini LEGO, pezzi di LEGO Technic (come ingranaggi, assi e parti pneumatiche) per costruire robot e altri sistemi automatici e/o interattivi.

LEGO Mindstorms é stato distribuito commercialmente come "RIS" (*Robotic Invention System*), é stato anche venduto ed utilizzato come sistema educativo, originalmente attraverso una partnership con il MIT. Può essere usato per costruire un modello di sistema integrato con parti elettromeccaniche controllate da computer. Praticamente tutti i tipi di sistemi integrati elettromeccanici esistenti nella vita reale (come gli elevatori o i robots industriali) possono essere modellati con i Mindstorms.

La prima generazione di Lego Mindstorms era costruita intorno ad un mattone programmabile conosciuto come RCX. Esso conteneva un microcontroller Renesas H8/300 come CPU interna e veniva programmato scaricando il programma (che poteva essere scritto in vari linguaggi di programmazione) da un PC o da un Macintosh sulla sua RAM attraverso una speciale interfaccia ad infrarossi.

Il nuovo prodotto Mindstorms é il Mindstorms NXT, che é stato rilasciato ad agosto 2006. Il kit comprende tre servomotori (molto piú grandi di quelli contenuti nella "vecchia confezione"), un sensore tattile, un sensore luminoso, un nuovo sensore sonoro, un sensore di prossimitá (a ultrasuoni) e un nuovo mattoncino intelligente NXT. Oltre alle migliorie apportate ai componenti del kit, il mattoncino intelligente NXT é stato dotato di porta bluetooth la quale può essere utilizzata non solo per caricare eventuali applicazione sul robot, ma anche per comandare la macchina da remoto.

3.1 Panoramica sulla tecnologia NXT



Figura 3.1: *Kit Mindstorms NXT*.

1. L' **NXT** é il cervello dei robot Mindstorms. É un mattoncino computerizzato ed intelligente che rende possibile l'esecuzione di tutti i compiti che un robot deve eseguire. Le caratteristiche tecniche sono le seguenti:
 - microcontrollore a 32-bit ARM7.
 - coprocessore 8 bit Atmel ATmega48 (classe AVR: è un RISC a 8 bit) a 8 MHz, con 4k flash e 512 byte RAM.
 - 256KB memoria flash, 64KB RAM.
 - Interfaccia bluetooth v2.0+EDR (chipset CSR BlueCore 4 version 2, clockato a 26 MHz, con propri buffer RAM e firmware stack Bluelab 3.2) velocità teorica massima 0,46 Mbit/sec (per trasferire il software o per controllare il robot da remoto).
 - porta USB full speed (12 Mbit/s).
 - Display LCD bianco e nero da 100?64 pixel.
 - Speaker mono 8 bit fino a 16 KHz.
 - 4 porte di input (riservate ai sensori).
 - 3 porte di output (riservate ai motori).
 - Alimentazione con 6 batterie AA (1.5V)) oppure tramite batteria ricaricabile al litio.

2. il **Sensore tattile** fornisce al robot il senso del tatto. Esso rileva quando viene premuto, e rilasciato. Può essere utilizzato per sapere se il robot ha qualche oggetto fra le mani da catturare.
3. il **Sensore del suono** è capace di catturare i decibel dei rumori nell'ambiente circostante. Può misurare livelli di pressione di suono fino a 90 db circa, per rendere di più semplice lettura tali misurazioni, esse vengono espresse in percentuale, alcuni esempi:
 - 4-5 % rumore all'interno di un soggiorno silenzioso.
 - 5-10% suono di conversazione nelle vicinanze.
 - 10-30% suono di conversazione vicino al microfono o musica suonata o normale livello.
 - 30-100% persone che gridano io musica ad alto livello.
4. il **Sensore di luce** è uno dei due sensori che danno la visione al robot (l'altro è il sensore ad ultrasuoni). Esso permette di distinguere tra luce e buio.
5. il **Sensore ad ultrasuoni** è forse il sensore più usato nella costruzione di robot. Grazie ad esso il robot è in grado di vedere e scoprire oggetti, lo si può usare per evitare ostacoli, misurare distanze o rilevare dei movimenti. Le distanze sono espresse in centimetri o in pollici, e vanno da un massimo di 255 a un minimo di 0 centimetri con una precisione di ± 3 centimetri. Per il suo funzionamento utilizza lo stesso principio scientifico usato dai pipistrelli: misura la distanza calcolando il tempo impiegato da un'onda sonora a colpire un oggetto e ritornare indietro, come un'eco. Il sensore è più sensibile a oggetti grandi con superfici dure, mentre lo è meno con oggetti realizzati in stoffa o curvati(come una palla), oppure molto sottili e piccoli.
6. il **Servomotore** è responsabile degli spostamenti, o in generale dei movimenti, della macchina. Ha al suo interno un sensore di rotazione per misurare sia la velocità che la distanza, i quali vengono trasmessi al processore del NXT. Vi è la possibilità di sincronizzare in maniera accurata i vari servomotori per poterli fare lavorare assieme.

Capitolo 4

Lego Mindstorm e ReSpect

4.1 Linguaggio di coordinazione

Un modello di coordinazione può essere visto come la colla che lega più attività assieme. Tale modello fornisce una struttura sulla quale entità attive ed indipendenti, chiamati *agenti*, possono svolgere i propri compiti. Le questioni di cui un modello di coordinazione dovrebbe occuparsi sono, la creazione, la distruzione e la collocazione spaziale degli agenti, nonché la distribuzione e la sincronizzazione delle loro azioni. Ma quali sono i componenti di un sistema di coordinazione?

- **Entità**, le cui interazioni sono gestite e controllate dal modello. Sono anche chiamate "*entità coordinabili*", possono essere processi unix-like, threads, oggetti concorrenti o anche utenti.
- **Media**, astrazione che regola e rende possibile le interazioni fra gli agenti. È il cuore, attorno al quale i componenti del sistema vengono organizzati. Alcuni esempi sono i semaphors, i monitors o i tuple spaces.
- **Regole di coordinazione**, le quali definiscono il comportamento del media, in risposta alle interazioni. Tali regole vengono utilizzate dagli agenti per capire come poter agire sul media di coordinazione e quali primitive poter utilizzare (es. in/out/rd...). Definiscono anche la struttura delle informazioni che verranno "manipolate" all'interno del media e dai vari agenti (es. tuples, elementi XML, oggetti Java...).

Modelli di coordinazione di questo genere sono solitamente sviluppati attorno al concetto di *spazio di dati condiviso*. Tale spazio

non è altro che un "contenitore indirizzabile" di strutture dati. Se più processi autonomi stanno collaborando in una stessa attività, essi non possono comunicare fra loro in maniera diretta, ma solamente affidandosi a tale spazio condiviso. Tali modelli, detti anche modelli *data-driven* sembrano essere migliori per applicazioni il cui numero di entità che devono collaborare non è a priori conosciuto.

4.1.1 Tuple-Space come meta modello

Il *tuple-space* è uno spazio di calcolo globale, all'interno del quale vengono elaborate delle strutture dati chiamate *tuple*. Le tuple sono una collezione ordinata, potenzialmente eterogenea, di informazioni, e rappresentano il linguaggio di comunicazione del nostro meta modello. Esse hanno vita proprio all'interno del tuple-space, sono ugualmente accessibili dagli agente, ma non sono collegate a nessuno di esse. L'accesso ad una tupla avviene in maniera associativa (controllando, cioè, il loro contenuto o struttura) piuttosto che per indirizzamento o naming.

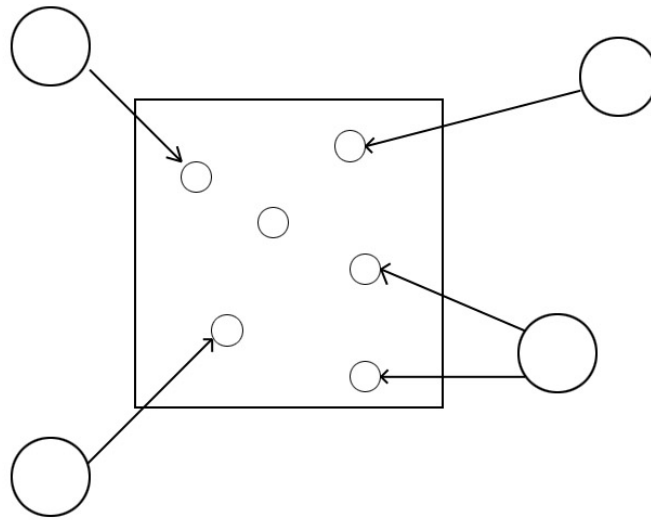
Il linguaggio di coordinazione, tramite il quale inserire e recuperare tuple dal tuple space è Linda. Esso utilizza come linguaggio di comunicazione:

- **Tuple**, es. `persona("nome",Luca), done(yes)` .
- **Template** (o Anti-Tuple) che identificano un set, una classe di tuple. Es. `persona("nome", X), done(Y)` .
- Meccanismo di **Tuple-Matching**, meccanismo tramite il quale si cerca di avere un match fra una tupla ed un template. Il match con una tupla avviene se: il numero dei campi è uguale, tipi, valori e lunghezza delle stringhe del template corrispondono a quelli della tupla, tipi e lunghezza dei campi formali corrispondono a quelli dei corrispondenti campi della tupla.

Linda è stato concepito con un set veramente ristretto, ma nello stesso tempo completo, di primitive:

- **in(TT)**, legge e toglie dal tuple-space la tupla che fa match con il template TT. Se più di una tupla fa match con il template, ne viene scelta una in maniera non deterministica. Se non è presente alcuna tupla facente match con il template proposto, l'operazione viene sospesa fino a quando non viene trovata la tupla opportuna.

- **out(T)** , inserisce nel tuple-space la tupla T.
- **rd(TT)** , legge dal tuple space, senza rimuoverla, la tupla che fa match con il template TT. Nel caso in cui non vengano trovate tuple facenti match con il template, o al contrario, ci siano più di una tupla corretta, il comportamento rimane identico a quello precedentemente descritto per la primitiva *in(TT)*.

Figura 4.1: *Tuple-space meta modello.*

Caratteristica di Linda, oltre alla coordinazione basata sul matching informativo, è quella di agevolare la comunicazione e la collaborazione trasversale fra gli agenti. Infatti, due agenti per comunicare non hanno la necessità di essere nello stesso ambiente nel medesimo momento, si parla in questo caso di *disaccoppiamento temporale e spaziale*.

4.1.2 Da Linda a ReSpect

Nota la struttura tuple-space precedentemente descritta, come potremmo risolvere il problema di eseguire un'attività dopo che, ad esempio, un certo numero di agenti ha fatto richiesta per una determinata risorsa? Più in generale, di cosa abbiamo bisogno per basare la nostra coordinazione su *eventi* che vengono scatenati piuttosto che sul cambiamento o meno delle informazioni?

È evidente che, con il modello precedentemente illustrato (tuple-space), il quale basa il meccanismo di coordinazione unicamente sulla presenza o assenza di dati, non è possibile realizzare tutti i pattern di coordinazione *control-driven* (es. *event-driven*). Ciò che è stato fatto è aggiungere uno strato di controllo control-driven su quello già esistente di Linda, facendo bene attenzione a non aggiungere ulteriori primitive.

Viene così introdotto il concetto di *tuple-center* (centro di tuple): tuple-space unito ad un automa a stati finiti che effettua computazioni event-driven. Possiamo immaginare che, ogni operazione interna al tuple centre, in entrata o in uscita, generi un evento che viene catturato dall'infrastruttura. L'automa a stati finiti, interno al tuple-centre, effettua le computazioni necessarie al fine di eseguire particolari azioni (che possono modificare, o meno, lo stato interno del tuple centre) in base al tipo di evento catturato. Insieme con l'introduzione del concetto di tuple centre nasce *ReSpecT* (Reaction Specification Tuples) come linguaggio di programmazione di tuple-center. ReSpecT può essere visto come estensione di Linda e offre la possibilità di specificare reazioni : azioni da eseguire in base al verificarsi di particolari eventi, definiti nella sintassi del linguaggio, che modificano lo stato del tuple centre. Le reazioni in ReSpecT sono del tipo $reaction(E, G, R)$, dove:

- E** rappresenta il template con il quale l'evento che viene a verificarsi deve fare match per poter triggerare la reaction corrispondente.
- G** è la guardia (possono esserne specificate anche più di una), rappresenta la condizione che deve essere verificata dopo che l'evento E è stato catturato.
- R** rappresenta la lista delle reazioni che devono essere eseguite (in termini di codice), dopo che la reazione è stata triggerata.

Ciò che mancava inizialmente a ReSpecT era la possibilità di specificare, in termini di reazioni, predicati di coordinamento relativi ad eventi ambientali, riguardanti risorse esterne. Le numerose modifiche ed estensioni al linguaggio ReSpecT hanno portato verso la così detta *Situatedness*.

ReSpecT Situatedness, con l'introduzione di alcune modifiche alla sintassi originale, consente di catturare eventi spazio-tempo ed ambientali, e di specificare vere e proprie *reazioni ambientali*. Sarà tale estensione di ReSpecT il linguaggio di coordinazione sul quale baseremo il nostro sistema di controllo per robot mobili. Qui di seguito viene riportata la nuova semantica:

- $\langle SimpleTCEvent \rangle$ (rappresentano tutti i possibili eventi accettati dal tuple centre) è stato esteso con $\langle EnvPredicate \rangle$ che si traduce in:

$\langle EnvPredicate \rangle ::= getEnv(\langle Key \rangle, \langle Value \rangle) | setEnv(\langle Key \rangle, \langle Value \rangle).$

Questo rende il TC in grado di accettare eventi ambientali, requisito base per il nostro sistema di controllo. Un evento di tipo $getEnv(., .)$ viene solitamente generato da un sensore, e rappresenta una lettura di informazioni sull'ambiente esterno. $setEnv(., .)$, invece, viene solitamente generato da un attuatore per indicare una modifica apportata sull'ambiente.

- $\langle TCPredicate \rangle$ (rappresentano tutte le possibili azioni eseguibili nel corpo di una reazione) è stato esteso con $\langle TCEnvPredicate \rangle$ che si traduce in:

$\langle TCEnvPredicate \rangle ::= \langle EnvResIdentifier \rangle ? \langle EnvPredicate \rangle$
dove: $\langle EnvResIdentifier \rangle ::= \langle EnvResName \rangle @ \langle NetworkLocation \rangle.$

In questa maniera il TC è in grado, all'interno del corpo di una reazione, di interrogare risorse ambientali per rilevazioni o attuazioni.

- $\langle ObservationPredicate \rangle$ (rappresentano quei predicati utilizzati per recuperare informazioni sull'evento corrente) è stato esteso con $env(\langle Key \rangle, \langle Value \rangle)$, predicato che permette di recuperare, data una chiave, il valore di una proprietà del particolare evento ambientale. Tale primitiva è stata introdotta considerando gli eventi ambientali, molto più ricchi di informazioni rispetto a quelli ordinari (generati dall'interazione con processi/agenti).
- L'insieme delle guardie 'e stato esteso introducendo $from_env$ e to_env . Le guardie sono condizioni che devono verificarsi al fine di eseguire il corpo della reazione. Le due nuove guardie vengono utilizzate per testare se la sorgente o la destinazione dell'evento è una risorsa ambientale.

Anche l'automa a stati finiti del tuple-center viene estesa, con l'introduzione di una coda di eventi ambiente $EnvQ$ e l'aggiunta di uno stato $fetchEnv$ nel quale si ha l'aggiunta a Re di tutte le reazioni che corrispondono ad un determinato evento ambientale.

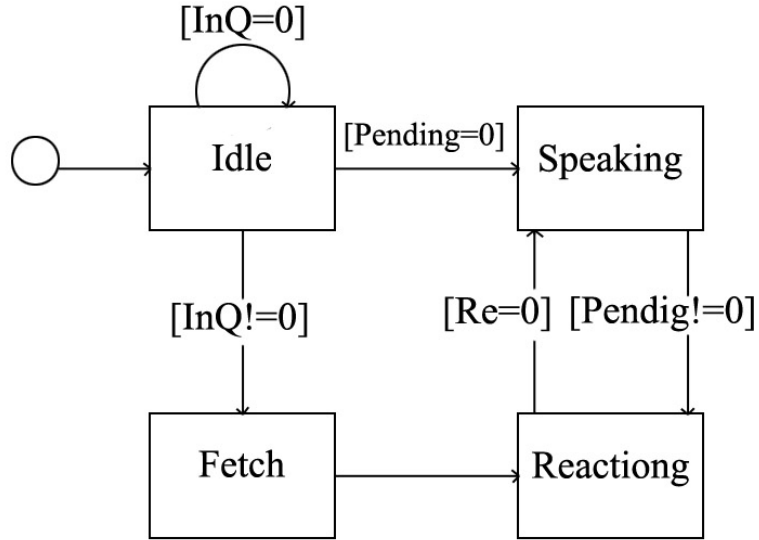


Figura 4.2: *Automa a stati finiti interna al tuple-space.*

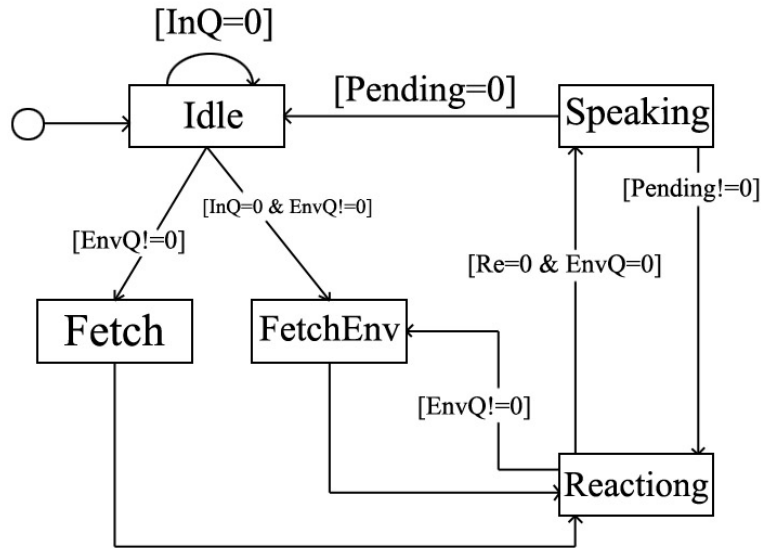


Figura 4.3: *Automa a stati finiti modificato per la situatedness.*

4.2 Architettura basata su ReSpecT per la Situatedness

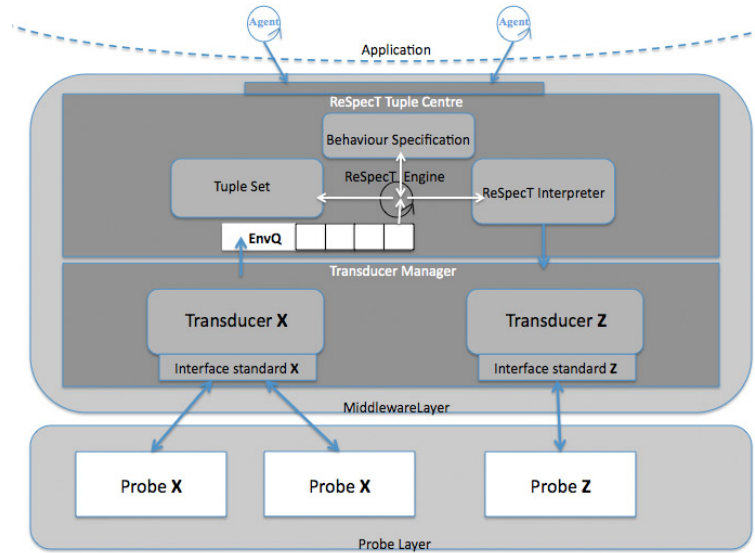


Figura 4.4: *Architettura per la situatedness.*

Oltre che fornire ReSpecT di un supporto per la situatedness a livello di sintassi, dobbiamo anche realizzare un'architettura concreta, a livello di framework, che possa supportare tale linguaggio. Questo è necessario per consentire la comunicazione con i dispositivi fisici, quali attuatori e sensori, più in generale, per poter tradurre eventi ambientali in eventi ReSpecT mediante la semantica descritta nel paragrafo precedente.

Per soddisfare questo requisito, la prima cosa da fare è fornire il ReSpecT framework di un layer composto da entità software chiamati *transducer*, il cui compito è quello di nascondere i dettagli tecnici della comunicazione fra sensori/attuatori e tuple-center, provvedendo a realizzare una rappresentazione uniforme degli aventi ambientali. Nella figura 4.4 è descritta l'architettura per il supporto alla situatedness.

Il *Prob* layer, implementa i servizi necessari alla comunicazione con i sensori e attuatori in accordo con gli standard adottati dalle tecnologie. Esso, infatti, fa uso di API software fornite dai costruttori hardware. Il probe layer si prende cura della gestione delle risorse fisica a basso livello, fornendo un strato software uniforme ed indipendente dalle specifiche hardware. In parole povere, grazie alle API

il prob layer è in grado di ricevere/inviare informazioni dai/a sensori ed attuatori fisici. In particolare, i ReSpect tuple-center vengono provvisti di un *transducer*, uno per ogni standard hardware. I trasducer vengono, a loro volta, gestiti da un altro componente software chiamato *transducer manager*. I trasducer incapsulano le politiche necessarie per comunicare con il prob layer; essi ricevono i dati dal livello sottostante sfruttando le API software, e hanno il compito di trasformarle in astrazioni ReSpect-like che verranno successivamente inserite nella *coda degli eventi ambientali EnvQ* di cui ogni tuple-center è stato fornito. Quest'ultimo compito è svolto dal *transducer manager*. Esso riceve informazioni dal *transducer*, le quali sono indipendenti dalla tecnologia hardware da cui sono stati percepiti, genera un evento ambientale che viene poi, come già detto in precedenza, inserito nella coda ambientale del tuple-center. Ogni *transducer manager*, inoltre, tiene traccia, di tutte le risorse ambientali che interagiscono con un determinato tuple-center, memorizzandosi un identificatore della risorsa (o meglio, del trasducer che rappresenta la risorsa), e un identificatore del tuple-center stesso. In questa maniera, un *transducer manager* non solo è in grado di gestire più di una risorsa (quindi più di un trasducer), ma consente anche la comunicazione bidirezionale fra l'architettura e l'ambiente.

Quando un predicato *set_env/get_env* viene eseguito all'interno del body di una reaction, un nuovo evento di output viene generato dal *transducer interpreter* ed inviato al *transducer manager* il quale lo invierà al *transducer* corretto che sospenderà la sua esecuzione fino a quando non avrà una risposta. Si può quindi notare che i componenti *transducer* hanno due comportamenti, il primo li vede attendere passivamente informazioni provenienti dall'ambiente esterno, l'altro, invece, li vede inviare informazioni ai componenti fisici dopo che una reaction interna è stata eseguita.

Una possibile alternativa sarebbe stata quella di eliminare la figura del *transducer manager*. Questa soluzione, però, porterebbe a definire il *transducer* come un componente costretto a sapersi interfacciare sia con le risorse esterne, che con l'architettura interna del sistema. Avremmo in definitiva componenti più "pesanti" e "complessi", sia dal punto di vista funzionale che implementativo, si introdurrebbe una *centralizzazione* nella gestione dei *transducer* con una conseguente diminuzione di robustezza del sistema, diminuzione di prestazione (sovraccaricamento) e maggiori difficoltà nel mantenere la "mappatura" delle risorse (tuple-center e sensori/attuatori) che comunicano fra di loro.

D'altro canto invece, la soluzione proposta inizialmente, non solo

presenta meccanismo di gestione delle comunicazioni fra risorse più liscio, ma dà anche la possibilità di aggiungere (o rimuovere) trasducer in maniera conveniente e dinamica in modo da poter testare l'infrastruttura con più standard possibili.

4.3 Un MAS per controllare un robot fisico

In questo paragrafo adotteremo ReSpect Situated per realizzare un MAS (Multy-Agent System) basato sull'architettura ad inclusione proposta da Brooks e descritta precedentemente nel capitolo 2. Tale modello ci permette di realizzare un controllo software per un insieme di componenti hardware soggetti a cambiamenti dinamici, più in generale, ogni qualvolta l'ambiente esterno subisce una modifica, è necessario che il sistema reagisca eseguendo operazioni opportune. Come esempio applicativo focalizzeremo la nostra attenzione sul caso di un reale robot mobile collocato in un generico ambiente (una stanza, un luogo aperto, un laboratorio ecc.) nel quale potersi muove in maniera autonomo, e in qualche modo, interagire con l'ambiente esterno che lo circonda. L'obiettivo che vogliamo raggiungere per il nostro robot è quello di farlo vagare in maniera non del tutto casuale, evitando gli ostacoli che gli si propongono durante il tragitto, proprio come descritto per il robot fisico proposto da Brooks.

Il nostro sistema di controllo, quindi, sarà strutturato in livelli, ognuno dei quali, indipendente dall'altro, implementerà un comportamento che vogliamo attribuire al robot fisico (nel nostro caso avremo due livelli di competenza, il primo che avrà l'obiettivo di evitare gli oggetti, il secondo di fare girare il robot nell'ambiente). Quando vorremo aumentare i "comportamenti" del nostro robot, ci basterà aggiungere nuovi livelli su quelli già esistenti. Il risultato finale sarà una struttura il cui flusso di controllo rimane completamente decentralizzato, e le comunicazioni (e quindi la coordinazione) fra i vari moduli dei livello, avviene tramite soppressione o inibizione dei segnali rispettivamente di ingresso o di uscita. Inoltre, il nostro robot non avrà una rappresentazione simbolica dell'ambiente esterno, ne tanto meno ne realizzerà una mappatura, semplicemente ogni modulo sarà in grado di reagire alle informazioni provenienti dai sensori.

Come risultato finale ci troveremo ad avere ciascun livello composto da agenti e tuple-center. Gli agenti verranno impiegati per i compiti che richiedono di prendere decisioni in maniera attiva, basandosi sui

dati presi dal mondo reale, mentre i tuple-center verranno utilizzati per reagire ai cambiamenti provenienti dall'esterno. In più, i tuple-center incapsuleranno i sensori e gli attuatori, in questo modo, le percezioni, provenienti dai sensori che andranno poi a comandare gli attuatori, saranno più facilmente elaborate in maniera simboli dal linguaggio.

L'architettura che realizzeremo verrà usata per controllare, da remoto, tramite tecnologia bluetooth, un robot fisico basato sulla tecnologia Lego Mindstorms. Tuttavia, l'architettura ha un obiettivo più ampio, che è quello di definire un nuovo approccio per la realizzazione di sistemi pervasivi basati sul paradigma dei multy-agent system.

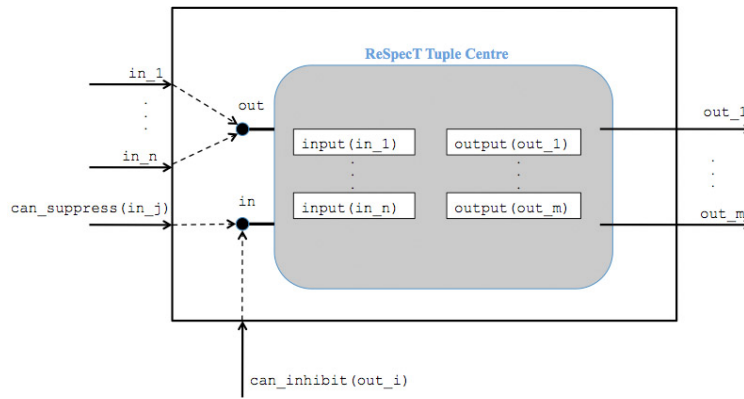


Figura 4.5: *Architettura interna di un modulo concepito come ReSpecT tuple-center.*

Qui di seguito andremo a descrivere, in maniera più chiara ed approfondita, come il meccanismo delle inibizioni e soppressioni dei segnali può essere modellato in ReSpecT. La figura 4.5 mostra la struttura di un singolo modulo rappresentato da un tuple-center. I segnali di input di un modulo sono rappresentati da operazioni di *out* eseguite da altri tuple-center o agenti, mentre i segnali di output di un modulo continuano ad essere rappresentati da operazioni di *out* eseguite su tuple-center esterni. Nel caso in cui l'output sia destinato ad un agente, esso sarà in grado di catturare tale informazioni in maniera attiva, eseguendo una operazione di *in/ inp* sul teuple-center in questione. Ogni modulo memorizza al suo interno una tupla *input(Id)* per ogni agente o tuple-center connesso alle proprie linee di ingresso. Il termine *Id* è l' identificatore che rappresenta la risorsa, ad esempio se la tupla *input(tc1)* è presente all'interno di un modulo, significa che al tuple-center *tc1* è consentito inviare dati a tale modulo. Analogamente, la tupla *output(Id)*, specifica quale tuple-center o agente può ricevere da-

ti dal modulo. Ad esempio se la tupla $output(tc2)$ è presente, questo significa che il modulo può eseguire sul tuple-center $tc3$ una qualsiasi operazione di output. Nel caso venga specificato un agente anziché un tuple-center, mediante la presenza della tupla $outputagent1$, sarà l'agente stesso ad eseguire una operazione di *in* per ricevere i dati.

Mediante le due tuple, $input(Id)$ e $output(Id)$, è possibile realizzare il meccanismo di soppressione o inibizione dei segnali. Per ogni linea di ingresso che potrebbe essere soppressa, il tuple-center ha un tupla del tipo $can_suppress(In1, Tcs, Time)$, dove $In1$ rappresenta l' identificatore del tuple-center la cui linea di connessione a questo modulo deve essere soppressa, Tcs è l' identificatore del tuple-center responsabile della soppressione e $Time$ è un valore numerico che esprime per quanti millisecondi l'input di $In1$ rimarrà soppresso. Come accadeva per l'architettura di Brooks, i dati provenienti da $In1$ vengono rimpiazzati con quelli di Tcs .

Ad esempio, se siamo in presenza della tupla $can_suppress(tci, tcs, 5000)$, significa che il tuple-center tcs può sopprimere i dati di input al modulo provenienti da tci per un tempo di 5 secondi. Quando il tuple-center tcs vorrà sopprimere tci non dovrà fare altro che eseguire una operazione di *in* sul modulo specificando la tupla $can_suppress(tci, tcs, 5000)$. Il modulo in questione reagirà con la seguente reazione:

```
- reaction( in(can_suppress(Tci,Tcs,Time)), (exo, response),
  ( in(input(Tci)), out(input(Tcs)),current_time(CurrentT),
    NewTime is CurrentT + Time,
    out_s( reaction(time(NewTime), endo,
      ( in(input(Tcs)), out(input(Tci)),
        out(can_suppress(Tci,Tcs,Time)))))).
```

L'operazione di *in* eseguita da tcs farà scattare la suddetta reazione quando la tupla $can_suppress(tci, tcs, 5000)$ verrà ottenuta da tcs stesso. Questa reazione semplicemente va a rimpiazzare gli input provenienti da tci con quelli provenienti da tcs , eseguendo le due operazioni di *in* e *out*, e va ad inserire una nuova reazione nel set delle reaction del tuple-center la quale scatterà dopo 5 secondi ed avrà il compito di ristabilire le linee di input originali e consentire, reinserendo la tupla $can_suppress(tci, tcs, 5000)$, una futura soppressione di tci da parte di tcs . Mentre è in atto la soppressione del segnale, tcs non può fare altre richieste di soppressione di tci , e il modulo può ricevere

dati in ingresso solamente da *tci* e dalle altre linee di ingresso non sopprresse.

Il tuple-center (o l'agente) che vuole inviare i dati al modulo eseguirà una operazione di *out* inserendo la tupla *data(X)*, dove *X* rappresenta l'informazione che si vuole inviare. Il modulo reagirà con le due seguenti reazioni:

- reaction(out(data(X)), (exo, response),
 (event_source(Tc), rd(input(Tc)),
 in(data(X)), out(X))).
- reaction(out(data(X)), (exo, response),
 (event_source(Tc), no(input(Tc)), in(data(X)))).

La prima avrà successo quando l'operazione di read (*rd(input(Tc))*) andrà a buon fine, in questo caso l'informazione che il tuple-center voleva spedire al modulo sarà inserita con successo; diversamente invece, se l'operazione di *out* sul modulo è eseguita da un tuple-center che in quel momento risulta soppresso (quindi la tupla *input(Tc)* non è presente), la tupla che contiene i dati viene immediatamente tolta grazie alla seconda reazione.

Per quanto riguarda il metodo di inibizione, che viene applicato per gli output di un modulo (mentre ricordiamo che la soppressione viene fatta per gli input), si opera in maniera analoga. Per ogni linea potenzialmente da inibire il modulo ha una tupla del tipo *can_inhibit(Out, Tci, Time)*, dove *Out* rappresenta il tuple-center al quale il modulo è connesso, *Tci* l' identificatore del tuple-center inibitore, e *Time* è il valore in millisecondi che rappresenta la durata dell'inibizione del segnale.

Ad esempio, una tupla del tipo *can_inhibit(tco, tcs, 2000)* significa che l'output sul tuple-center *tco* viene inibito per 2 secondi dal tuple-center *tcs*. Come abbiamo già descritto in precedenza, *tcs* per inibire *tco* esegue una operazione *in* sul modulo, facendo richiesta della tupla *can_inhibit(tco, tcs, 2000)*. Il modulo a sua volta reagirà con la seguente reazione, del tutto simile a quella già proposta per la soppressione:

- reaction(in(can_inhibit(Tco,Tcs,Time)), (exo, response),
 (in(output(Tco)), current_time(CurrentT), NewTime

```

is CurrentT + Time, out_s( reaction(time(NewTime),
                                endo, ( out(output(Tco)), out(can_inhibit(Tco,Tcs,Time)))))).

```

Quando *tcs* vuole inibire un output, la precedente reazione scatta; essa andrà semplicemente a rimuovere *tco* dal set delle tuple autorizzata a ricevere informazioni dal modulo, e andrà anche ad inserire una nuova reaction, la quale scatterà dopo 2 secondi, e servirà a reinserire il tuple-center precedentemente inibito. La differenza rispetto al caso descritto per la soppressione è che il segnale inibito questa volta non viene sostituito con nessun altro segnale. Quindi, quando un modulo vuole inviare dati ad un tuple-center, semplicemente controllerà se il target desiderato è presente nelle tuple di output, questo viene fatto eseguendo un comando di *rd*; se tale comando fallisce (cosa che tipicamente avviene quando una tupla *out* è temporaneamente tolta a causa di una richiesta di inibizione) i dati non vengono inviati, diversamente il modulo può eseguire un'operazione di output sul tuple-center desiderato. Come già detto, quando il destinatario è un agente, non è il modulo ad eseguire una *out* su di esso, bensì è l'agente stesso a procurarsi i dati eseguendo una *inp(data(X))*, dove X rappresenta le informazioni desiderate. A questo punto il modulo reagisce con la seguente reazione:

```

- reaction(inp(data(X)), (exo, request),
           ( event_source(Ag), rd(output(Ag)), in(X), out(X))).

```

Questa reazione semplicemente controllo se l'agente in questione è autorizzato a ricevere i dati, in caso affermativo il modulo spedisce i dati richiesti.

4.3.1 Accesso alle risorse ambientali

Un altro aspetto fondamentale da prendere in considerazione per il nostro sistema di controllo è come poter modellare, mediante ReSpect Situated, delle entità che ci consentano di mappare le informazioni provenienti dal mondo esterno. Tale compito è lasciato ai così detti *wrapper* delle risorse ambientali e viene realizzato con dei tuple-center, in questo modo, tale risorse potranno essere manipolare e gestite semplicemente usando le già note primitive ReSpect. Le interazioni con il mondo esterno sono rese possibili grazie alle primitive *setEnv* e *getEnv*,

rispettivamente usate per leggere informazioni dai sensori, e inviare comandi agli attuatori. La reale comunicazione che c'è fra i *wrapper* e le specifiche risorse esterne è coadiuvato dai *transducer*, i quali sono gli unici ad essere in grado di tradurre i protocolli di comunicazione con le tecnologie esterne e mapparli nella semantica delle primitive *setEnv* e *getEnv*. Nella figura 4.6 è riportata la struttura di un tuple-center usato da wrapper per un sensore ad ultrasuoni.

Come si può notare dalla figura, quando viene eseguita una nuova lettura dal sensore, corrispondente ad una nuova percezione dell'ambiente esterno, il tuple-center reagisce al template *set_env(Key, Value)*, dove *Key* e *Value* devono essere rimpiazzati con il nome della specifica proprietà misurata, e il corrispondente valore letto, ad esempio, nel nostro caso il sensore ad ultrasuoni sarà usato per la lettura delle distanze degli oggetti, quindi è abbastanza logica utilizzare una sintassi del tipo *set_env(distance, 20)*. Da notare che la vecchia lettura viene sempre eliminata prima che quella nuova venga inserita nel tuple-center. Ogni modulo collegato ai *wrapper* potrà di fatto accedere ai sensori semplicemente eseguendo una *in(reading(Key, Value))* sul wrapper corretto.

Nella figura 4.7 invece, viene rappresentato un *wrapper* per un attuatore, che nel nostro caso corrisponde ad uno dei due motori responsabili del movimento del robot. Quando un modulo interno necessita di comandare un attuatore, sarà sufficiente che esegua, sul wrapper, una operazione di *out* della tupla *command(Key, Value)*, dove *Key* va sostituito con la proprietà da comandare, e *Value* con il valore corrispondente. Ad esempio, nel nostro caso, potremmo voler comandare la potenza a cui gira il motore, questo sarà possibile semplicemente emettendo la tupla *command(power, 80)*, con la quale si vuole settare la potenza del motore al suo 80%. Il wrapper reagisce all'inserimento di un comando con la reazione descritta in figura che, dopo aver rimosso la tupla *command*, invia la richiesta all'attuatore mediante la tupla *set_env(Key, Value)*.

4.3.2 Architettura del MAS basata su ReSpecT

Basandosi sulle tecnologie e metodologie fino ad ora proposte, andiamo a realizzare un Multy-Agent System basato su ReSpecT che ci consenta di comandare, da remoto, un robot mobile costruito con un kit della Lego Mindstorm, grazie al quale è possibile realizzare un robot autonomo, completo in tutte le sue parti, in maniera economica e rapida. Il cuore del nostro robot è il brick NXT, un micro-computer

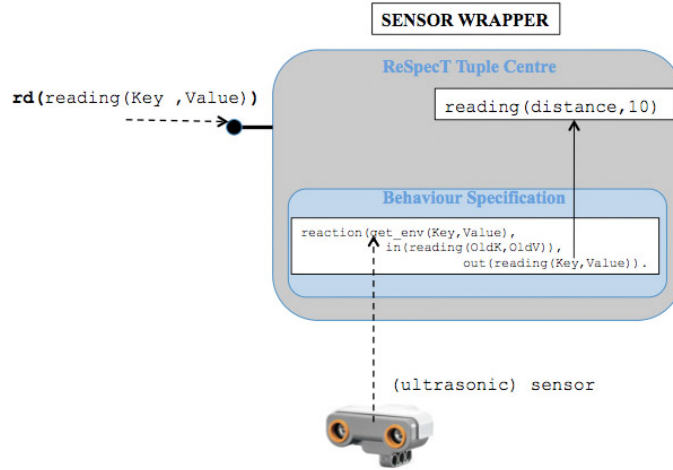


Figura 4.6: *ReSpec tuple-center* usato da wrapper per un sensore ad ultrasuoni.

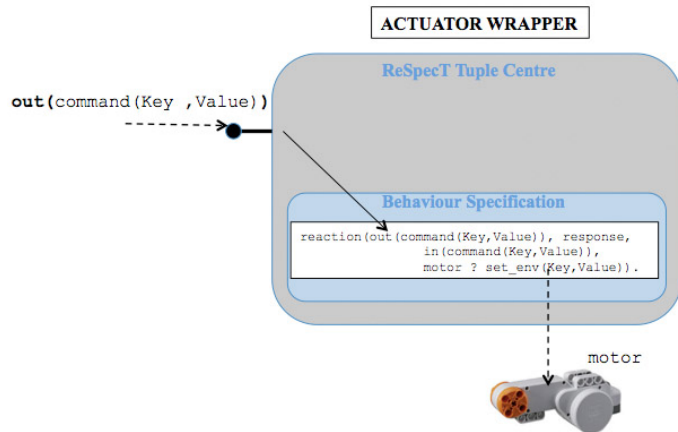


Figura 4.7: *ReSpec tuple-center* usato da wrapper per un attuatore.

dotato anche di tecnologia bluetooth. Grazie a questa tecnologia sarà possibile interagire con il nostro robot da remoto, inviando dati agli attuatori e ricevendo informazioni dai sensori. Per fare ciò ci avvarremo della tecnologia ICommand. Questo è un package Java grazie al quale potremmo interagire con il brick NXT utilizzando codice Java. Fino a quando ReSpect sarà sviluppato in Java, ICommand continuerà ad essere una valida e logica soluzione alla comunicazione a distanza con il robot.

Il robot realizzato (visibile nella sezione "Esempio applicativo") è equipaggiato con tutti i sensori a disposizione nel kit, ma quello che verrà utilizzato per i nostri esperimenti sarà solamente quello ad ultrasuoni. Il robot è provvisto anche di due ruote, ognuna delle quali comandata da un servo motore elettrico, che può essere controllato, così da consentirci di settare velocità ed angolazione in cui si muoverà il nostro robot. Il sensore ad ultrasuoni posizionato nella parte superiore del robot ci consentirà di identificare eventuali oggetti presenti nell'ambiente, e di calcolarne la loro distanza. Per interagire mediante ReSpect con il robot fisico, abbiamo costruito due tipi di trasducer: *ultrasonicTrasducer* che ci consente di ricevere le distanze degli oggetti dal sensore ad ultrasuoni, e *motorTrasducer*, per controllare i servo motori del robot, entrambi sono implementati come classi Java. La comunicazione con i sensori e gli attuatori fisici viene basata sulle API di ICommand.

Il nostro sistema sarà quindi composto da un *ultrasonicTrasducer* e due *motorTrasducer* (uno per ogni servo motore), entrambi non comunicano direttamente con il MAS Layer, ma vengono gestiti dai *trasducerManager*, componenti implementati sempre come classe Java. Il compito dei *trasducerManager* è quello di interfacciarsi direttamente con il MAS Layer, in particolare, dopo aver ricevuto le informazioni, ReSpect-Like, dai *trasducer*, essi inseriscono i nuovi eventi ambientali nella coda degli eventi dei tuple-center, adibiti da wrapper, se si tratta di letture dei sensori, diversamente, quando i *trasducerManager* lavorano con attuatori, ricevono eventi interni generati dal MAS Layer e li indirizzano verso l'esterno al giusto *trasducer*. Come si può intuire, i *trasducer* e i *trasducerManager* sono due componenti con caratteristiche implementative e funzionali completamente diversi ed indipendenti; infatti un *trasducer* può lavorare sotto qualsiasi tipo di architettura, sia essa ReSpect-Based o meno, tutto quello che sa fare è leggere o inviare informazioni alle risorse esterne. I *trasducerManager*, invece, devono conoscere con esattezza il funzionamento dell'architettura con cui essi devono interfacciarsi, mentre sono completamente

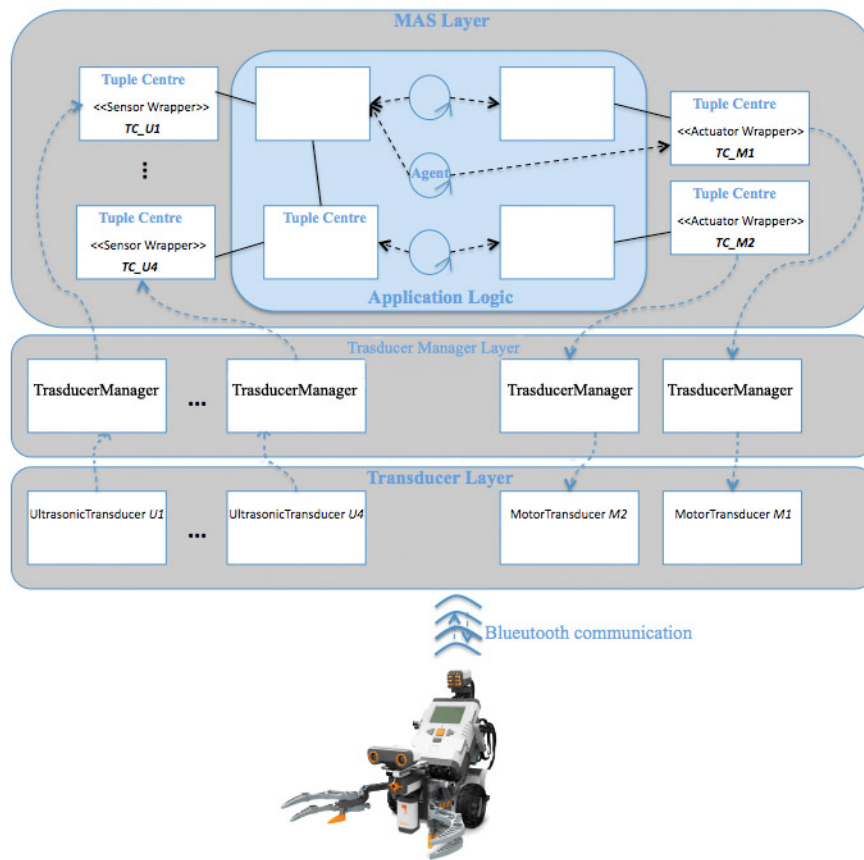


Figura 4.8: Architettura logica per un MAS basato su ReSpect per controllare un robot mobile da remoto.

indipendenti dal tipo di risorsa esterna utilizzata.

Una possibile alternativa, sarebbe stata quella di utilizzare un solo *trasducerManager* per tutti i *trasducer* presenti. Questa soluzione, nonostante sia facilmente realizzabile grazie alla struttura interna dei *trasducerManager*, secondo il nostro punto di vista introduce diversi svantaggi. Per prima cosa, si centralizzerebbe tutto il lavoro dei *trasducer*, con lo svantaggio di ridurre la robustezza dell'intero sistema, infatti, in presenza di un mal funzionamento del *trasducerManager*, tutto il sistema ne risentirebbe. Ancora, utilizzare un solo *trasducerManager* introdurrebbero rallentamenti nell'interazione con il mondo esterno; infatti, il numero dei *trasducer* da gestire può aumentare in maniera dinamica e senza preavviso costringendo il *trasducerManager* ad un sovraccarico di lavoro.

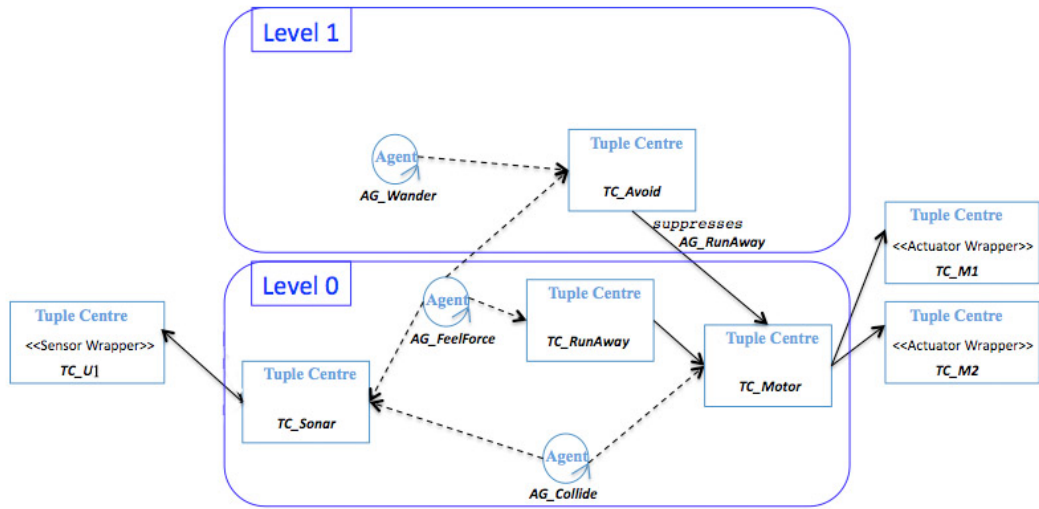


Figura 4.9: Architettura logica interna per il MAS basato su ReSpec.

La figura mostra come il sistema di controllo si componga di tre livelli principali, il *trasducer Layer*, contenente i *trasducer* per la comunicazione con il robot fisico, il *TrasducerManager Layer*, nel quale abbiamo i vari *trasducerManager* che realizzando il bind fra *trasducer* e wrapper, e in fine abbiamo il textitMAS Layer il quale contiene tutti i componenti del nostro MAS, incluso i wrapper, tutti i tuple-center interni, gli agenti e le applicazioni logiche. La figura 4.9 rappresenta il sistema di controllo proposto da Brooks espresso in termini di tuple-center e agenti al posto dei moduli. Tale sistema implementa i primi due livelli di competenza, il Livello 0 responsabile di fare evitare

gli ostacoli al robot, e il Livello 1, che renderà il robot in grado di aggirarsi in maniera autonoma per l'ambiente.

4.3.3 Livello 0: evitare gli ostacoli

Questo livello si occupa di fare evitare gli ostacoli al robot, siano essi in movimento che fissi. Esso è composto da tre tuple-center e due agenti.

- **TC_Sonar** tuple-center è responsabile della lettura delle distanze degli oggetti tramite il *TC_U1 wrapper*, in questo modo può generare l'informazione che rappresenta la distanza degli oggetti "frontali" al robot. Questo tuple-center è provvisto delle seguenti reaction:
 - reaction(time(1000), (endo),
 (tc_u1 ? rd(reading(distance,X)), out(done))).
 - reaction(out(done), (endo),
 (in(done), current_time(Time), NewTime
 is Time + 1000 out_s(reaction(time(NewTime),
 (endo),(tc_u1 ? rd(reading(distance,X)),out(done)))))).
 - reaction(rd(reading(distance,X)), (exo),
 (current_source(tc_u1), out(distance(front,X)))).

Dopo aver effettuato la lettura della distanza grazie alla prima reaction, la seconda viene triggerata. Questa inserisce una nuova reaction che dopo un tempo pari a 1 secondo scatterà automaticamente ed eseguirà una nuova lettura dal wrapper. Grazie alla *time reaction* possiamo continuare ad eseguire letture in maniera automatica e periodica nel tempo. Ogni lettura provoca la reazione della terza reaction, la quale non fa altro che tradurre la distanza letta dal wrapper, specificando a quale direzione la distanza percepita fa riferimento (nel nostro caso, il sonar rileva oggetti solamente nella direzione frontale al movimento del robot, con un'angolazione totale di 60 gradi).

- l'agente **AG_Collide** è uno dei due agenti autorizzati a recuperare informazioni dal TC_Sonar tuple-center. Esso, eseguendo una operazione di *inp* (non bloccante), richiede la tupla *distance(front,value)* grazie alla può monitorare la presenza di ostacoli. Il semplice algoritmo che esegue è il seguente:

1. *TC_Sonar ? inp(data(distance(front,X)))*
2. *if X < 15cm then TC_Motor ? out(data(halt))*
3. *ritornare al punto uno.*

L'agente ripetitivamente esegue una *inp* sul TC_Sonar. Se un oggetto si trova ad una distanza inferiore ai 15 cm dal robot , viene inviata una tupla di *halt* al TC_Motor il quale provvederà immediatamente a fermare i motori.

- L'agente **AG_Feelforce** è il secondo agente autorizzato alla lettura di informazioni direttamente dal TC_Sonar tuple-center. Il suo compito è quello di calcolare la velocità a cui devono essere settati i motori e la direzione verso la quale il robot deve muoversi. I valori vengono calcolati in base alle distanze e la collocazione degli ostacoli percepiti dal sensore ad ultrasuoni e consente al robot di continuare a muoversi evitando gli ostacoli. L'algoritmo eseguito è il seguente:

1. *TC_Sonar ? inp(data(distance(front,X)))*
2. *computeDirection(X, Angle, Speed)*
3. *TC_RunAway ? out(data(direction(Angle,Speed)))*
4. *TC_Avoid ? out(data(direction(Angle,Speed)))*
5. *ritornare al punto uno*

i due valori *Angle* e *Speed* rappresentano rispettivamente la direzione che il robot deve prendere, espressa in gradi (fra 0 e 360), e la velocità a cui esso deve muoversi, espresso come numero naturale compreso fra 0 (fermo) e 100 (velocità massima). *computeDirection(X, Angle, Speed)* è stato realizzato, per comodità, come metodo interno all'agente, ma poteva anche essere realizzato come predicato prolog. É al suo interno che viene sviluppato l'algoritmo per calcolare in maniera ottimale la nuova velocità e direzione. Una volta ottenuti i due valori essi verranno indirizzati ai tuple-center adeguati. Da notare che, se il Livello 1 non viene realizzato, il passo 4 può essere eliminato.

- Il terzo tuple-center che andiamo ad analizzare è il **TC_RunAway**, il quale, periodicamente, riceve informazione dall'agente *FeelForce*. La reazione di cui è fornito è la seguente:

```

– reaction( out(direction(Angle, Speed)), (endo),
            ( in(direction(Angle, Speed)), rd(threshold(Th)),
              Speed >= Th, tc_motor ? out(data(command(Angle,Speed))))
          ).

```

Tale reaction scatta ogni qual volta viene inserita una nuova tupla del tipo *direction(Angle, Speed)*, se il nuovo valore di *Speed* supera o eguaglia il valore di una prefissata soglia *Th*, allora la reazione avrà successo e la tupla *command(Angle, Speed)* verrà passata al TC_Motor. Diversamente significa che la nuova velocità non produce un cambiamento "importante" all'andamento del robot, per questo la tupla verrà ignorata.

- L'ultimo tuple-center che compone il Livello 0 è il **TC_Motor** il quale, semplicemente, trasforma i comandi ricevuti da FeelForce, in comandi per i due motori del robot, wrappati dai due tuple-center TC_M1 e TC_M2. Nel caso in cui il tuple-center riceva un *halt* da Collide esso reagirà immediatamente con la seguente reazione:

```

– reaction( out(halt), (endo),
            ( tc_m1 ? out(command(power,0)),
              tc_m2 ? out(command(power,0)) ) ).

```

grazie alla quale i due motori verranno fermati. D'altra parte, invece, quando viene ricevuta una tupla del tipo *command(Angle,Speed)* la reazione che viene triggerata sarà la seguente:

```

– reaction( out(command(Angle,Speed)), (endo),
            (tc_m1 ? out(command(angle,Angle)),
              M2_angle is - Angle, tc_m2 ? out(command(angle,M2_angle)),
              tc_m1 ? out(command(power,Speed)),
              tc_m2 ? out(command(power,Speed)))) ).

```

Questa reazione si preoccupa di inviare ai wrapper dei motori le velocità a cui devono essere settati i servo motori, così come la direzione espressa in gradi. È importante notare, in questo caso, che gli l'angolo inviato ai due motori sono uno l'opposto dell'altro, per permettere al robot di ruotare in maniera corretta.

4.3.4 Livello 1: muoversi nell'ambiente

Questo livello rende il robot capace di muoversi nell'ambiente senza urtare oggetti; per raggiungere tale scopo deve forzatamente collaborare con il Livello 0. Esso è composto da un tuple-center e un agente.

- L'agente **AG_Wander** seleziona una nuova direzione e una nuova velocità ogni n secondi, e successivamente invia la tupla *random_direction(Angle, Speed)* all'unico tuple-center del Livello 1, ovvero TC_Avoid.
- TC_Avoid riceve la tupla *random_direction(Angle, Speed)* e *direction(Angle, Speed)* rispettivamente da AG_Wander e AG_FeelForce. Il suo compito è di fare una sorta di "mix perfetto" fra le velocità e le direzioni proposte dai due agenti, sempre avendo come obiettivo base quello di fare muovere il robot nell'ambiente senza farlo urtare contro oggetti. Le reaction di cui è fornito il tuple-center sono:

- `reaction(random_direction(Angle, Speed)), (response),
(in(random_direction(Angle,Speed), in(direction(Angle,Speed))))`
- `reaction(out(direction(A, S)),(response),
(in(direction(A,S)),in(random_direction(Angle,Speed)),
out(data(Angle,Speed,A,S))))`.
- `reaction(out(data(Angle,Speed,A,S)),(endo),
(select_direction(Angle,Speed,A,S,X,Y),
out(selected_direction(X,Y)),(tc_motor ? in(can_suppress(tc_runaway,
tc_avoid,3000))))`.

La terza reaction, la quale contiene nel proprio body, il predicato prolog *select_direction(Angle, Speed, A, S, X, Y)*, mediante la quale viene calcolata la direzione e la velocità per il robot, deve scattare solamente quando le due tuple, dei due agenti precedentemente descritti, sono entrambe presenti nel tuple-center. Questo controllo viene realizzato mediante le prime due reaction. Come ultimo compito, TC_Avoid sopprime l'input di TC_RunAway a TC_Motor eseguendo l'operazione *in(can suppress(tc_runaway,tc_avoid,5000))* il cui funzionamento è stato descritto nel paragrafo 4.3.

TC_Avoid, una volta ottenuta la tupla *can_suppress(tc_runaway,tc_avoid,3000)* reagisce con la seguente reazione:

– reaction(in(can_suppress(tc_runaway,tc_avoid,5000)),
 (response, exo), (in(selected_direction(Ao,So)),
 tc_motor ? out(command(Ao,So))).

Questa reazione semplicemente ottiene la tupla richiesta e invia la tupla *command(A0, S0)* al TC_Motor il quale a sua volta provvederà ad inviarla ai wrapper dei servo motori.

Capitolo 5

Esempio applicativo



Figura 5.1: *Robots mobili usati per testare l'architettura.*

L'architettura proposta e descritta in questa tesi, risulta essere funzionale e reattiva agli stimoli esterni. L'implementazione del sistema di controllo [7] è stata testata con i due robots in figura. Nonostante i ritardi di comunicazione dovuti a vincoli fisici (bluetooth, frequenza di lettura dei sensori, elaborazione dati dei tuple-center) i due tester si sono dimostrati capaci di aggirarsi autonomamente negli ambienti proposti, evitando gli eventuali ostacoli.

Bibliografia

- [1] Rodney A. Brooks: A.I memo 864. *A robust layer ed control system fo a mobile robot*. September 1985
- [2] Matteo Casadei e Andrea Omicini: *Programming Tools for Engineering Agent-Environment Interaction in ReSpecT*.
- [3] Matteo Casadei e Andrea Omicini: *Situated Tuple Centres in ReSpecT*.
- [4] Andrea Tosi: *Trasduttori e midleware, da JDDAC alla coordinazione*. 2008
- [5] Matteo Casadei e Andrea Omicini: *Situating AA ReSpecT for Pervasive Environment Applications*. 2008-2009
- [6] Elena Nardi e Andrea Omicini: *Coordination in Multi-agent Systems*. 20 novembre 2008
- [7] Maicol Urbinati: *Tecnologie di coordinazione per la situatedness in ambito robotico*.