

ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Seconda Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

STUDIO E SPERIMENTAZIONE DI MODELLI
DI INTERAZIONE E COORDINAZIONE
SOFTWARE DI ROBOT LEGO MINDSTORMS

Elaborata nel corso di: Sistemi Operativi LA

Tesi di Laurea di:
ALBERTO PINI

Relatore:
Prof. ALESSANDRO RICCI

Correlatore:
Ing. MATTEO CASADEI

ANNO ACCADEMICO 2007–2008
SESSIONE I

PAROLE CHIAVE

Lego Mindstorms

Modelli di comunicazione di alto livello

Comunicazione basata su scambio di messaggi

Dedicato ai miei genitori, a mia sorella, e a tutti i
miei amici e compagni di facoltà che mi hanno
aiutato e sostenuto in tutti questi anni. Un
ringraziamento particolare a Maurizio.

Indice

Introduzione	xi
1 LEGO Mindstorms NXT	1
1.1 Panoramica	1
1.2 Hardware	2
1.2.1 Input	4
1.2.2 Output	5
1.2.3 Altre periferiche	6
1.3 Software e linguaggi di programmazione	6
1.3.1 NXC (Not eXactly C)	7
1.3.2 iCommand	9
1.4 Impiego dei robot LEGO Mindstorms	10
1.4.1 LEGO Mindstorms in ambiente universitario e di ricerca	11
1.4.2 Altri progetti	16
1.4.3 Considerazioni	19
2 Supporto alla comunicazione	21
2.1 La comunicazione tra Brick NXT	21
2.2 Modello di interazione e gestione delle connessioni di base tra più robot	22
2.2.1 Limiti del modello di comunicazione di base	22
2.3 Il protocollo di comunicazione	23
2.3.1 Limiti del supporto Bluetooth nativo	25
2.4 Soluzioni per migliorare il protocollo	27
2.5 Altre considerazioni	27

3	Modelli di comunicazione di alto livello	31
3.1	Introduzione	31
3.2	Modelli di comunicazione diretta	32
3.2.1	Scambio di messaggi diretto	32
3.3	Modelli di comunicazione indiretta	34
3.3.1	Scambio di messaggi indiretto	34
3.3.2	Modelli basati su spazi	36
3.4	Confronto fra le due tipologie di comunicazione	37
3.5	Considerazioni	38
4	API per la comunicazione diretta	39
4.1	Obiettivo	39
4.2	Le nuove API per la comunicazione diretta	41
4.3	Architettura della libreria	43
4.3.1	La struttura	43
4.3.2	Il mediatore	46
4.4	Implementazione	47
4.5	Deployment	50
4.5.1	Utilizzo del software	50
4.5.2	Operazioni preliminari	52
4.6	Collaudo	53
4.6.1	La libreria	53
4.6.2	Coordinazione	55
4.6.3	Performance	56
4.6.4	Motori e sensori	59
4.6.5	Problemi riscontrati	61
5	Casi di studio	63
5.1	Introduzione	63
5.2	Interazione fra robot: coordinazione nei movimenti	64
5.2.1	Risultati ottenuti	65
5.2.2	Considerazioni	66
5.3	Interazione fra robot: esplorazione	67
5.3.1	Comportamento dei robot	68
5.3.2	Considerazioni	69

6	Conclusioni	71
6.1	Sviluppi futuri	72
6.1.1	Miglioramento dell'affidabilità	72
6.1.2	Comunicazione indiretta	73
A	RCX	75
A.1	La prima generazione di mattoncino intelligente	75
A.2	Tipi di sensori	76
A.3	Altri componenti forniti	77
A.4	Linguaggi di programmazione	77
B	Linguaggi di programmazione per NXT	79
B.1	Caratteristiche	79
B.2	Performance	81
B.3	Controllo remoto	82
C	Tipologie di robot NXT	83
C.1	TriBot	83
C.2	RoboArm T-56	84
C.3	Spike	85
C.4	Alpha Rex	86

Introduzione

I LEGO Mindstorms sono sempre più utilizzati in ambienti universitari e di ricerca come kit di robot programmabili a basso costo per sperimentazioni in vari ambiti, in particolare Intelligenza Artificiale, Intelligenza Artificiale Distribuita e Sistemi Multi-Agente situati. In tali contesti sono di particolare interesse - tra le altre - le sperimentazioni che concernono sistemi formati da più robot che cooperano: questi esperimenti sono molto utili -ad esempio- quando si vogliono studiare e poi simulare sistemi distribuiti autonomi che devono collaborare tra loro per raggiungere un obiettivo finale.

Per tali sperimentazioni, tuttavia, il protocollo di comunicazione di base fornito dal kit, basato su tecnologia Bluetooth, presenta limiti significativi in termini di flessibilità, in particolare qualora si vogliano realizzare sistemi multi-robot con strategie di comunicazione e coordinazione di alto livello non banali. Questo perché basato su una architettura Master-Slave, per cui in generale non può esservi comunicazione diretta fra robot - in termini peer-to-peer: i robot ‘Slave’ possono comunicare sempre e solo con un robot ‘Master’, definito staticamente. Il protocollo nativo di comunicazione, quindi, non appare del tutto adeguato per utilizzarli come prototipi per esperimenti in cui si richiede l’impiego di più robot che collaborino insieme secondo strategie diverse da quella master-slave.

A partire dall’analisi di questo problema, la tesi ha come obiettivo l’introduzione di un supporto tecnologico software che estenda il supporto di base master-slave per rendere agevole per i progettisti e programmatori di sistemi multi-robot la realizzazione di modelli di comunicazione e coordinazione di più alto livello. In particolare il supporto è dato da una libreria scritta nel linguaggio NXC, fra i linguaggi di riferimento per LEGO Mindstorms NXT, con opportune API (Ap-

plication Programming Interface) che abilitano lo scambio di messaggi diretto, peer-to-peer, fra i robot.

Nella tesi si descrivono i passi fondamentali che hanno caratterizzato la realizzazione della libreria, dallo studio dei modelli di comunicazione di alto livello alla progettazione delle API, all'implementazione e quindi al collaudo. Per testare l'efficacia dell'approccio sono stati infine ideati e sviluppati due casi di studio, caratterizzati dallo sviluppo di due sistemi multi-robot con strategie di comunicazione e coordinazione di diversa complessità.

La tesi si compone di 6 capitoli:

- Il primo capitolo introduce i robot LEGO Mindstorms NXT e ne viene fatta una panoramica generale, spiegando cosa siano, come sono fatti, i tipi di linguaggi con cui programmarli ed aggiungendo anche alcuni riferimenti a progetti già realizzati;
- Il secondo capitolo si sofferma sull'analisi del modello adottato dai robot per comunicare, basato su tecnologia Bluetooth, spiegandone le politiche ed i numerosi limiti;
- Il terzo mira a studiare i principali e più diffusi modelli di comunicazione diretti ed indiretti, in particolare il Message Passing e lo Spazio di Tuple;
- Il quarto fornisce una soluzione concreta al miglioramento del modello di base, grazie ad una libreria che mette a disposizione nuove API; in questo capitolo è molto importante la sezione riguardante i test che hanno accompagnato tutta la fase della sperimentazione;
- Il quinto vuole proporre alcuni particolari casi di studio in cui i robot si devono coordinare per raggiungere un obiettivo comune, sfruttando le API appena realizzate;
- Nel sesto si traggono le conclusioni, si propongono dei miglioramenti sul lavoro eseguito ed infine si pensa ad alcuni possibili sviluppi futuri.

Capitolo 1

LEGO Mindstorms NXT

In questo capitolo si dà un quadro generale dei robot LEGO Mindstorms.

1.1 Panoramica

Il progetto Mindstorms è nato alla fine degli anni '80 al *MIT* (Massachusetts Institute of Technology) di Boston nell'ambito di una ricerca di orientamento costruttiva per avviare i più giovani al computer, integrando l'informatica a prodotti ad essi dedicati.

Nel 1998, la ditta danese della LEGO, nell'ambito di questo progetto, rilasciò l'*RCX*¹, un 'mattoncino intelligente' programmabile, dotato di collegamenti per sensori e motori, da applicare alle costruzioni. Nel 2006 la LEGO ha messo sul mercato una linea di prodotti molto più evoluta: la serie *NXT*.

Partita come un'idea dedicata ai più giovani, le potenzialità di questo prodotto hanno permesso che i Mindstorms fossero sfruttati anche in ambito della ricerca ed in settori molto più professionali tanto che sarebbe sbagliato considerarli un semplice giocattolo.

¹Appendice A - RCX

1.2 Hardware

La serie LEGO Mindstorms NXT permette di costruire in maniera veloce piccoli robottini dotati di un computer centrale, denominato *NXT Brick* (o più familiarmente ‘mattoncino intelligente’), in grado di memorizzare ed eseguire programmi, emettere suoni, collegarsi ad altri dispositivi, elaborare dati provenienti dall’ambiente esterno e controllare dispositivi in uscita. Partendo da questo indispensabile blocco base, verranno aggiunti tutti i pezzi restanti per comporre il robot ed ad esso si potranno collegare input ed output tramite cavi: se le porte non dovessero bastare è possibile mettere in comunicazione più Bick per farli lavorare in collaborazione via Bluetooth. Rispetto all’RCX, risulta avere una velocità di calcolo ed una capacità di memoria notevolmente maggiori, inoltre sono state migliorate ed aggiunte numerose funzionalità.



Figura 1.1: *NXT Brick*

Le principali caratteristiche tecniche di questo piccolo computer sono:

- Processore a 32 bit Atmel AT91SAM7S256 (classe ARM7) a 48 MHz
- 256 KB di Memoria Flash
- 64 KB di Memoria RAM
- Coprocessore a 8 bit Atmel ATmega48 (classe AVR: RISC a 8 bit) a 8 MHz
- 4 KB di Memoria Flash
- 512 Byte di Memoria RAM

Connettività:

- USB 2.0 (12 Mbit/s)
- Interfaccia Bluetooth V2.0+EDR (chipset CSR BlueCore 4 version 2, clockato a 26 MHz, firmware stack Bluelab 3.2, 47 KByte di propri buffer RAM interni, 8 MBit di memoria Flash esterna)
- 4 porte di input (cavo a 6 fili RJ12, con trasmissione digitale low-speed a 9600 bit/s con protocollo I2C, 1 porta high-speed a 1 Mbit/s IEC 61158 Type 4/EN 50170 compliant)
- 3 porte di output (cavo a 6 fili RJ12, con trasmissione digitale)



Figura 1.2: Interno NXT Brick:[1]Bluetooth, [2]Processore, [3]Coprocessore

Altre caratteristiche:

- Tastiera con 4 tasti in gomma
- Alimentazione: 6 batterie AA (1.5 V) o batteria ricaricabile Li-on da 1400 mAh
- Display LCD bianco e nero da 100x64 pixel (26x40.6 mm)
- Speaker a 8 kHz (canale sonoro mono a 8-bit e 2-16 KHz di frequenza di campionamento)

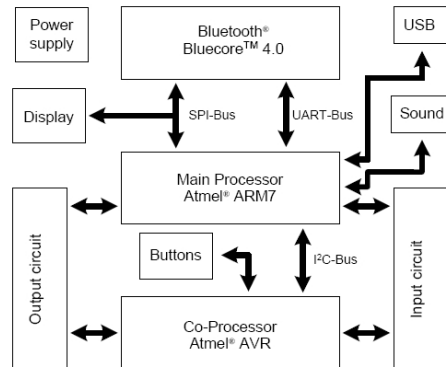


Figura 1.3: Interazioni fra le varie parti del Brick

1.2.1 Input

È possibile collegare fino ad un massimo di 4 sensori all'NXT Brick per permettere al robot di avere una percezione dell'ambiente esterno. Le tipologie di sensori sono:

- *Tatto*: rileva la pressione del pulsante posto all'esterno



Figura 1.4: Sensore di Tatto

- *Suono*: misura l'intensità sonora (il rumore) dell'ambiente



Figura 1.5: Sensore di Suono

- *Luce*: percepisce l'intensità luminosa ambientale o di un oggetto (vede il colore come una scala di grigi, il risultato è una tonalità che va dal chiaro allo scuro). È dotato anche di una luce led rossa che, riflettendo sulla superficie, ne fa assimilare la reale luminosità.

Figura 1.6: *Sensore di Luce*

- *Distanza*: è un emettitore di ultrasuoni. In base al tempo impiegato dall'onda per ritornare indietro dopo aver incontrato un ostacolo, calcola la distanza. È quindi possibile utilizzarlo anche come sensore di movimento.

Figura 1.7: *Sensore ad Ultrasuoni*

1.2.2 Output

L'NXT Brick è dotato di 3 porte a cui si possono collegare altrettanti *servo motori*, sincronizzabili tra loro.

Figura 1.8: *Servo motore*

Tramite quest'ultimi sarà possibile far muovere il robot, anche in risposta ai dati prelevati dall'interazione con l'ambiente per mezzo dei dispositivi di input, se programmato a dovere.

Sono dotati di sensori interni per misurarne la velocità, distanza percorsa e potenza.

1.2.3 Altre periferiche

Vi sono inoltre altri tipi di sensori più precisi, della serie HiTech (ad esempio ne esiste uno simile a quello di luce, che però riesce a riconoscere direttamente i colori invece di vedere solo gradazioni di grigio).



Figura 1.9: *Sensore di Colore*

Si possono collegare anche sensori della vecchia linea RCX (come quello di temperatura e di rotazione).

Inoltre è possibile inserire come output anche dei piccoli LED, oltre ai motori, tuttavia si prenderanno in considerazione solamente i componenti forniti nel kit base.

1.3 Software e linguaggi di programmazione

I robot sono molto versatili dal punto di vista della programmazione: si può ricorrere a vari tipi di linguaggi.

Vi sono ambienti di sviluppo interamente grafici, come NXT-G e Robolab, dove si può stabilire il comportamento dei robot tramite blocchi ed icone. Un'altra applicazione basata su componenti grafici è National Instruments LabVIEW (in collaborazione con la Tufts University), usato per misurazioni e testing di applicazioni a cui è possibile installare un apposito toolkit per LEGO NXT. Essendo interamente

grafici, si prestano per essere utilizzati da utenti non esperti o che vogliono realizzare agevolmente veloci applicazioni. Tra i linguaggi più tradizionali, invece, si ha a disposizione l' NBC (Next Byte Codes), in assembly, di più basso livello perchè legato al controllo diretto dell'hardware; linguaggi di alto livello come leJOS NXJ per programmare in Java; si può utilizzare Lua con pbLua o il classico C con RoboC e NXC (Not eXactly C).

In base alla scelta di linguaggio, ovviamente si avrà una ripercussione per quel che riguarda velocità, flessibilità e potenzialità di sviluppo, occupazione di risorse (memoria) e rapidità di esecuzione.²

In questa tesi si prenderà in considerazione l'NXC.

1.3.1 NXC (Not eXactly C)

L'NXC deriva dal NQC (Not Quite C) utilizzato dal precedente Brick RCX.

Ha una sintassi molto simile a quella del C, ma abbastanza più semplificata.

Con l'NXC, il comportamento di un robot viene programmato in termini di task, ossia attività che il robot deve svolgere.

Le primitive che fornisce sono specifiche per il controllo e l'uso di questi particolari robot.

Le API (Application Programming Interface), che definiscono costanti, valori, macro e funzioni, messe a disposizione dall'NXC possono essere raggruppate in varie tipologie:

- *Carattere generale*
Per operazioni che richiedono l'uso del timer interno; per il controllo del flusso del programma (inclusi i semafori e la gestione di più task); per usare e manipolare le stringhe; per gli array; per le funzioni matematiche
- *Modulo per l'input*
Tipi, modi d'uso e settaggio dei sensori; per prelevare informazioni da un sensore

²Appendice B - Linguaggi di programmazione per NXT

- *Modulo per l'output*
Per il controllo e la sincronizzazione dei motori
- *IO Map Address*
Per leggere e scrivere valori dei campi degli input ed output usando costanti di basso livello
- *Suono*
Per far emettere al Brick NXT una sequenza di toni o per riprodurre un file musicale tramite gli speaker interni
- *Modulo IOCtrl*
Per la comunicazione a basso livello dei due processori che controllano l'NXT (soprattutto per operazioni di spegnimento e aggiornamento)
- *Display*
Per poter disegnare linee, far comparire scritte o numeri sull'LCD monocromatico presente sul Brick
- *Loader*
Per la manipolazione del file system (creazione, apertura e altre operazioni sui file) e la gestione della memoria
- *Comando*
Per l'esecuzione dei programmi con la virtual machine dell'NXT
- *Pulsanti*
Per attribuire funzioni ai quattro pulsanti presenti sull'esterno del Brick NXT e per controllarli
- *Modulo UI*
Per vari aspetti dell'interfaccia utente per il Brick NXT (volume dello speaker, batteria, flag, stato del Bluetooth, Virtual Machine, USB, programmi utente presenti all'interno)
- *Modulo LowSpeed*
Per la comunicazione e gestione dei sensori digitali LowSpeed, cioè che usano il protocollo I2C

- *Modulo Comm*

Per la comunicazione e gestione dei sensori digitali HighSpeed e per altri tipi di protocolli di trasferimento ad alta velocità come USB e Bluetooth (scambio di messaggi, stato del trasferimento e della connessione)

- *Modulo Hi-Tech*

Per input Hi-Tech, come il sensore di colore o per utilizzare funzioni avanzate per l'RCX e lo Scout³, o la porta IR per lo scambio di messaggi per questi ultimi due.

L'IDE di sviluppo utilizzato per realizzare e compilare programmi scritti in tale linguaggio è *Bricx Command Center*; permette di connettere il Brick NXT tramite Bluetooth o USB per scaricare il programma, ma può essere usato anche per comandare direttamente gli output del robot usando un joystick virtuale. Bricx Command Center al momento della compilazione si incarica di convertire il programma scritto in NXC, in NXT bytecode che verrà compreso ed eseguito dall'interprete del Brick NXT. Per il cambio del nome di uno specifico Brick e per l'aggiornamento del firmware (attualmente alla versione 1.05) si è fatto ricorso, invece, al software fornito da LEGO.

1.3.2 iCommand

La libreria iCommand permette di programmare aspetti della comunicazione in ambiente di sviluppo java, tuttavia viene compilato in modo da poter essere interpretato dal firmware standard degli NXT, come i programmi scritti in NXC: a differenza di leJOS NXJ, che è il linguaggio da cui deriva dal punto di vista della sintassi, iCommand non richiede un firmware apposito sul Brick NXT che riesca ad interpretare esclusivamente codice java.

Tuttavia questa libreria si occupa solo di aspetti riguardanti la comunicazione, quindi di norma per poter programmare i robot si ricorrerà all'NXC: per applicazioni che risiedono su pc e che tentano di comunicare con i Brick, l'idea di utilizzare le librerie iCommand, invece, è

³Appendice A - RCX

quasi una scelta obbligata, poichè l'NXC non ne dà la possibilità. Per poter poi sfruttare tali librerie occorre però un supporto aggiuntivo per la comunicazione a basso livello: le librerie *RXTX*.

1.4 Impiego dei robot LEGO Mindstorms

Come detto precedentemente, questi tipi particolari di robot realizzati con le costruzioni LEGO vengono utilizzati in molti campi e con differenti scopi. Per la loro versatilità possono essere sfruttati sia a basso livello, cioè da principianti nel settore informatico, come lo possono essere i più giovani, sia per applicazioni e sistemi complessi, impegnando anche gli utenti più esperti.

Possono essere individuate principalmente 3 categorie di impiego:

Didattico Per spiegare i principi dell'informatica e della robotica, educando i più giovani, che vedono la LEGO ancora come un giocattolo.

Amatoriale Si tratta soprattutto di studenti universitari o di programmatori che fuori dall'ambito scolastico si divertono a mettersi alla prova realizzando modelli di robot complessi.

Universitario È l'unico ambiente che ovviamente prenderemo in considerazione, perchè, servendosi dei Mindstorms, mira a fini pratici e non ludici: in ambito universitario i robot vengono utilizzati soprattutto per la ricerca, la simulazione e la sperimentazione.

Il rapporto tra il basso costo di questo prodotto e la tecnologia messa a disposizione più che accettabile lo ha reso molto ambito e preferibile rispetto ad altri tipi soluzioni. Inoltre la caratteristica di essere costituito da LEGO ne permette una facile e veloce costruzione, senza bisogno di essere degli esperti nel campo della meccanica e senza ricorrere a saldature o incastri di componenti metallici di alcun genere: questo permette di concentrarsi esclusivamente sul comportamento del 'cervello' del robot, ossia sulla parte software. Vengono utilizzati prevalentemente nell'ambito di Intelligenza Artificiale e Reti Neurali: è indispensabile infatti trovare dei prototipi su cui provare velocemente il software 'intelligente' che potrà essere esportato successivamente nei

robot veri e propri realizzati per sostituire l'uomo in compiti gravosi. Questo tipo di approccio tramite LEGO Mindstorms servirà per affrontare varie problematiche che ovviamente si incontreranno; basta pensare solamente che in un futuro si potranno utilizzare robot per salvataggi estremi senza mettere a repentaglio la vita del soccorritore o per esempio per raggiungere luoghi estremi in cui l'uomo non può arrivare (come tunnel sotterranei o privi d'aria). Occorre necessariamente che il robot sia in grado di interagire con l'ambiente circostante ed evolversi: deve capire da solo come comportarsi e non si deve limitare alle istruzioni fornite da un programma o dagli ordini impartiti dall'uomo, perchè in alcuni ambienti ostili sarà esso che dovrà adeguarsi da solo non conoscendo a priori il luogo, prendere decisioni ed avere quindi un comportamento non programmato ma autonomo.

È inutile sottolineare l'importanza del lavoro di squadra e quindi avere una serie di robot intelligenti, che collaborano tra loro, perchè raramente un singolo robot potrà avere tutti i requisiti per raggiungere un determinato obiettivo.

Tutto questo è necessario sperimentarlo in primo luogo in laboratori, con modelli veloci da costruire ed economici.

Ma, soprattutto, una caratteristica fondamentale che viene considerata in questa tesi è il fatto che sono in grado di simulare un qualsiasi sistema informatico o il comportamento di più sistemi informatici distribuiti che interagiscono tra loro.

1.4.1 LEGO Mindstorms in ambiente universitario e di ricerca

Progetti di sviluppo e simulazione hanno coinvolto numerose università, sia italiane che estere.

Molti di essi sono realizzati ancora con i vecchi modelli RCX, poichè gli NXT sono di più recente uscita, tuttavia questo non significa che siano meno validi dei nuovi progetti, sebbene l'hardware sia molto più limitato.

Di seguito verranno riportati alcuni esempi che metteranno in evidenza l'importanza dei LEGO Mindstorms nel campo della ricerca (si tratta maggiormente di progetti di tesi).

Al *Politecnico di Milano* all'interno del Corso della Conoscenza e Sistemi Esperti è stata realizzata con i Mindstorms RCX, una coppia di agenti robotici dalle limitate capacità sensoriali. Questi robot sono in grado di muoversi nello spazio con l'obiettivo di spostare un oggetto disposto a caso e di riportarlo verso il margine superiore dell'ambiente. Per poter raggiungere tale scopo, i robot dovranno essere in grado di comunicare tra loro tramite scambio di messaggi per riuscire a sincronizzarsi. Questo progetto ha il fine di analizzare e migliorare i comportamenti di un agente reattivo.

Un altro progetto realizzato con gli RCX è stato condotto all'*Università degli Studi di Genova*. Si propone di studiare la comunicazione con infrarossi fra robot Lego mediante LeJOS. A tal scopo è stato realizzato un ambiente in cui interagiscono 2 tipi di entità: preda e predatore. Ogni robot a seconda dello stato in cui si trova ('caccia' o 'vagare') si muove nello spazio o alla ricerca della preda, aumentando la velocità per effettuare l'inseguimento o aggirandosi liberamente.

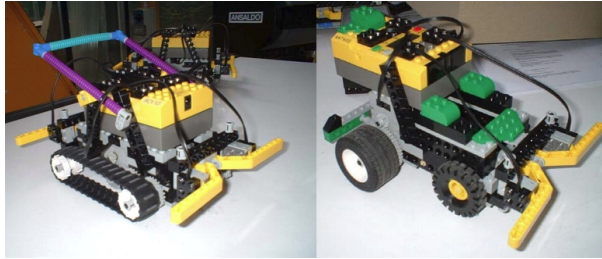


Figura 1.10: *Preda e predatore*

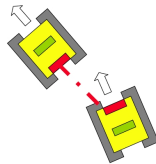


Figura 1.11: *Avvistamento*

All' *Università degli Studi dell'Aquila*, invece, nell'ambito dello studio dei Controlli Automatici, sono stati utilizzati i Mindstorms RCX come prototipi che siano rappresentativi dei sistemi presi in esame. Grazie ad essi si sono studiati per esempio tecniche di controllo avanzate come la modellazione nello spazio di stato e di tecniche tradizionali come il controllo proporzionale, integrale e derivativo (PID), sfruttando anche sensori ingegneristici, costruiti ad hoc. I Mindstorms sono stati quindi utilizzati per integrare questo tipo di studio e per elaborare dati forniti da sensori utili in questo campo.

Risulta essere interessante un progetto di simulazione di un diffrattometro condotto all' *Università di Bredley*. Una reale macchina di questo genere misura l'angolo di incidenza e di diffrazione di un raggio X. Questo esperimento mira a riprodurre questa sofisticata attrezzatura utilizzando un Brick NXT per l'elaborazione, un sensore di luce come rilevatore e un laser come sorgente.

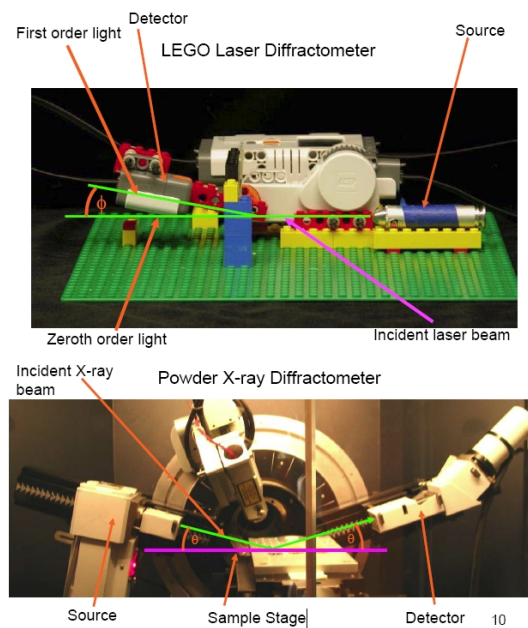


Figura 1.12: *Simulatore NXT di un diffrattometro*

Un progetto eseguito all' *Università degli Studi di Siena* prevedeva l'impiego di robot mobili per analizzare problemi dovuti alla competizione per l'ottenimento di risorse, alla cooperazione, all'organizzazione di gruppo, all'autoapprendimento ed al movimento coordinato, rendendosi conto che questo rappresenta un particolare sistema distribuito abbastanza complesso. I robot sono stati raggruppati in squadre mobili per il raggiungimento di un compito specifico determinato, mostrando così un comportamento cooperativo, allo scopo di migliorarne la qualità e la velocità di esecuzione.

Presso l' *Università di San Diego*, invece, è stato utilizzato un robot Mindstorms NXT per testare un nuovo tipo di sensore RFID (Radio Frequency Identification). Dopo aver interfacciato un lettore RFID al robot, quest'ultimo andrà in giro per l'ambiente a rilevare dati da schede magnetiche per poi elaborare i dati. Questo mette in luce il fatto di essere un prodotto versatile, in grado di adattarsi, di essere utile per qualsiasi tipo di esperimento e che l'hardware non si limita a quello fornito dalla LEGO ma è espandibile, potendo inserire dei propri sensori in grado di essere controllati dal Brick.

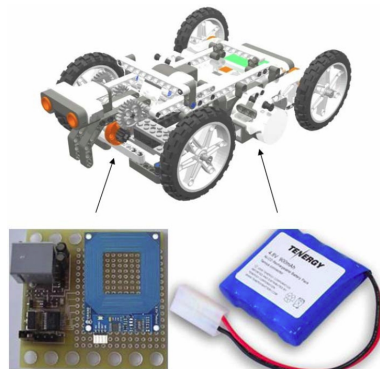


Figura 1.13: *RFID Reader applicato ad un NXT*

Al *DTU* (Technical University of Denmark) sono stati utilizzati per simulare un sistema multiagente: grazie ad esso si è in grado di sviluppare e studiare questo tipo di modello portandolo su dei robot che ne rispecchiano concretamente il comportamento e la comunicazione.

Questo ha permesso inoltre di implementare nuovi algoritmi software per risolvere problemi di base legati al movimento e per contribuire allo sviluppo di nuovi tipi di agenti più avanzati. Tutto questo permetterà di contribuire allo sviluppo di un software intelligente, molto più autonomo, in grado di reagire alle variazioni dell'ambiente circostante. Il progetto consiste nel realizzare 2 squadre caratterizzate da colori distinti: nero e argento. Essi si muovono lungo un percorso a forma di griglia; se un robot dovesse incontrare lungo il proprio cammino un ostacolo del suo stesso colore, allora lo rimuoverà, altrimenti comincerà ad un robot dell'altra specie dove poterlo trovare.

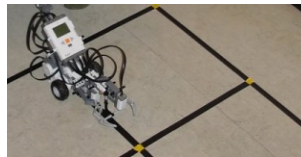


Figura 1.14: *Primo prototipo realizzato*

Presso l'*Università di Aarhus* (Danimarca) è stato realizzato un esperimento sulla base dei fondamenti di Reti Neurali. Non si vuole vedere un robot come una macchina, ma come un umano, in grado di reagire agli stimoli dell'ambiente e che, grazie alle percezioni sensoriali, sia capace di evolvere, di imparare e di adattare il proprio comportamento. Questo genere di studi sono molto importanti e costituiscono uno dei maggiori obiettivi che l'ingegneria informatica si impone ed affronta, cioè creare un software complesso dalla capacità di apprendere come il cervello umano.

All'*Università degli Studi di Modena e Reggio Emilia*, invece, è stata realizzata una rete wireless per la comunicazione di robot RCX; infatti questi ultimi dispongono della possibilità di comunicare tra loro solo tramite tecnologia IRDA. Il progetto si propone di far comunicare ogni RCX con la propria torretta di ricezione ad infrarossi la quale sarà collegata ad un palmare dotato di tecnologia wireless. Grazie a questo passaggio i vari palmari faranno da intermediari e da mezzi trasmissivi ad onde radio tra gli RCX, inoltre vi è anche l'ausilio di PC in grado

anch'essi di trasmettere e di ricevere via onde radio. Questo è un esempio di realizzazione di rete LAN composta da elementi e tecnologie eterogenee, che mette in evidenza i problemi che spesso questo tipo di ingegnerizzazione comporta.



Figura 1.15: Rete LAN Wireless

Anche l'*Università di Edinburgo* ha realizzato alcuni progetti con gli RCX, considerandoli sistemi multi-agente, con lo scopo di esplorare l'ambiente grazie alla cognizione sensitiva.

All'*Università 'Roma Tre'* si è fatto seguire ad un RCX una pista nera provvista di una serie di incroci; arrivato in prossimità di uno di essi, tramite il sensore di luce, riconosce se c'è una richiesta di una 'porta' di una camera alla sua destra, in caso affermativo ritorna quindi alla sua 'casa' prende un oggetto, fa di nuovo lo stesso percorso e riporta l'oggetto prelevato in quella stanza ed infine lo rilascia. Questo fa parte di un progetto di Intelligenza Artificiale e si propone anche di analizzare e migliorare il software per il controllo del robot.

1.4.2 Altri progetti

Un progetto di grande interesse è *Breedbot*, il robot da allevare. È stato creato dalla collaborazione tra il 'Laboratorio di vita artificiale e robotica' dell'*Istituto di Scienze e Tecnologie Cognitive (Istc)* del *Consiglio Nazionale delle Ricerche di Roma*, l'*Università Federico II di Napoli* e l'*Università degli Studi di Palermo*. L'esperimento consiste nel partire

con una serie di 9 robot ognuno dei quali si sposterà virtualmente in uno schermo ed avrà un proprio codice genetico di comportamento grazie al quale cercherà di adattarsi dentro ad un ambiente interamente software (all'interno di un PC). Ad ogni generazione l'utente sceglierà quali saranno i migliori per poi farli riprodurre e quindi verrà introdotta una seconda generazione di robot virtuali, e così via. Dopo una serie di passaggi di generazione si otterrà quindi un robot che sarà in grado di adattarsi meglio di tutti ad un ambiente ostile e che sarà capace di reagire meglio agli stimoli esterni. Si tratta dunque di far crescere un'intelligenza artificiale in un ambiente virtuale ed alla fine trasferirla all'interno di un robot reale. Lo studio può sembrare un semplice gioco, ma in futuro potrebbe avere importanti risvolti in diverse discipline scientifiche, come la ricerca spaziale, per esplorare ambienti estremi, come Marte: a tal scopo il robot deve essere il più autonomo possibile, senza che l'uomo intervenga per dirgli cosa fare, in modo tale da riuscire a far fronte anche agli imprevisti e per avere un software così intelligente occorre effettuare una selezione di programmi.

Vengono usati anche in settori tecnologici avanzati: per esempio la *Vernier Software & Technology* prendendo parte ad un progetto con LEGO Education, National Instruments, Tufts University e Carnegie Mellon University Robotics Academy ha testato i propri sensori su robottini Mindstorms NXT. La Vernier, infatti, produce ed elabora strumenti tecnologici per la misurazione compresi ad esempio sensori sofisticati per sostanze chimiche: tra quelli testati figurano quello di corrente (DCP-BTA), di salinità, del pH, di conduttività, di ossigeno liquido, di UV, della temperatura dell'acciaio inossidabile, del campo magnetico e l'accelerometro.

A livello amatoriale è stato realizzato un progetto da Daniele Beneditelli, studente universitario che come hobby realizza robot assai complessi. Non rientra nella ricerca universitaria, ma è molto significativo sotto molti punti di vista. Infatti mostra come sia possibile utilizzare gli NXT in modo tale da produrre e sviluppare un software intelligente e complesso. Si tratta del risolutore del classico cubo di Rubik: è dotato di una videocamera che trasferisce l'immagine ad un computer che la elabora, riconosce i colori dei singoli blocchi di una

faccia ed in base ad algoritmi risolutivi, aziona i motori del robot NXT in modo tale che i suoi bracci girino e ruotino le facce del cubo, fino a quando queste ultime non presenteranno ognuna un unico colore differente dalle altre.

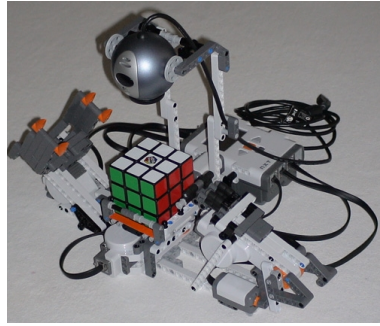


Figura 1.16: Risolutore del cubo di Rubik

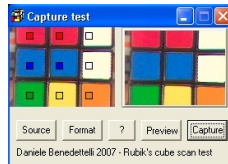


Figura 1.17: Riconoscimento dei colori

Sono poi spesso promosse iniziative dalla *First LEGO League* della LEGO Company nel Nord America ed in Europa (Scandinavia, Regno Unito, Germania). Sono gare in cui si mettono in competizione i robot realizzati a livello amatoriale. Apparentemente sembrano essere inutili e organizzate solo per puro intrattenimento tuttavia esse vengono sponsorizzate da industrie leader di molti settori e sostenute da Istituti di Ricerca rinomati ed all'avanguardia, sia privati che pubblici. Questo significa che possono servire per indurre e stimolare programmatori che partecipano alla competizione alla progettazione di sistemi evoluti per poi farne oggetto di studio o per poterne trarre spunto per sfruttarli in campi più pratici come quello industriale.

1.4.3 Considerazioni

Molti di questi progetti mettono soprattutto in risalto la necessità di creare software che cerchi di essere il più autonomo possibile e che riesca ad interagire con l'ambiente circostante e con altre entità.

Anche a livello di Intelligenza Artificiale non si vuole in modo particolare riprodurre la capacità di pensare della mente umana, ma di realizzare strumenti in grado di integrarsi nei sistemi già in uso. Per raggiungere tale scopo deve essere capace di mutare a partire dalle sue potenzialità originali grazie alle sue capacità sensoriali; a differenza dell'approccio tradizionale dell'informatica in cui un computer riceve in ingresso una serie di dati, li elabora e poi produce un risultato in uscita, nei sistemi robotici gli input sono percepiti continuamente e devono essere prodotte immediatamente delle risposte agli attuatori. Lo studio si concentra dunque anche su comportamenti emergenti, cioè non programmati, che emergono dall'interazione con un fenomeno e che possono comparire quando un numero di entità semplici operano in un ambiente, dando origine a comportamenti complessi di una collettività, che impara ad autorganizzarsi; robot progettati sulla base di Reti Neurali poi sono in grado anche quindi di evolversi da soli e di apprendere.

In questa tesi non si prenderà in considerazione la facoltà di apprendere dei robot, ma soltanto la capacità di coordinarsi tra loro ed eseguire comportamenti comuni: le osservazioni, i progetti e le finalità dei progetti precedentemente descritti sono solo una parentesi per illustrare l'importanza in campo informatico dei Mindstorms.

Capitolo 2

Supporto alla comunicazione

Questo capitolo descrive il metodo di comunicazione a distanza adottato dai robot NXT per interagire con altri device e fornisce un'analisi per appurarne i limiti.

2.1 La comunicazione tra Brick NXT

I vecchi robot, aventi come computer l'RCX, potevano adottare una comunicazione a distanza tramite porte ad infrarossi (IRDA). Questo tipo di tecnologia è molto limitata perchè presuppone una vista diretta (e senza ostacoli interposti) della porta con cui devono comunicare. Nei modelli NXT questo tipo di approccio è stato sostituito dal più moderno Bluetooth, che consente una ricezione ad una distanza massima di 10 metri con tecnologia ad onde radio, permettendo una connessione senza preoccuparsi di ostacoli o di visione diretta dell'altro dispositivo.

Un robot può comunicare con altri Brick NXT o con altri tipi di device dotati di ricezione Bluetooth, come un PC (con la possibilità di collegarlo anche via USB) un cellulare, un palmare, tramite i quali si può controllarlo impartendogli ordini che agiscono direttamente sui motori (utilizzando un software già predisposto per questa funzione). Sfruttando le caratteristiche del protocollo di base è possibile creare una rete Bluetooth formata al massimo da 4 dispositivi che, per il momento, considereremo rappresentati esclusivamente da robot.

2.2 Modello di interazione e gestione delle connessioni di base tra più robot

Come detto in precedenza è possibile connettere tramite Bluetooth un massimo di 4 robot in modo tale che comunichino tra di loro, tuttavia la gestione avviene in modo particolare: deve essere dichiarato un *Master* che avrà il ruolo di computer centrale e darà il via alla comunicazione. Ad esso, quindi, possono essere collegati i restanti robot (per un massimo di 3) che avranno la funzione di *Slave* ed accetteranno la connessione. Gli *Slave* (registrati alla linea 1, 2 o 3) possono comunicare solamente con il proprio Master (canale radio 0). Un robot non può avere il ruolo di Master e Slave allo stesso tempo.

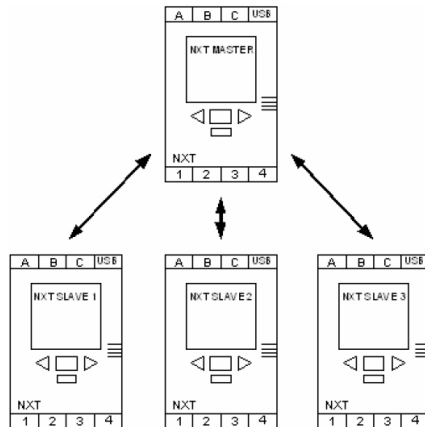


Figura 2.1: Comunicazione tra 4 NXT via Bluetooth

2.2.1 Limiti del modello di comunicazione di base

Questo modello si dimostra molto inadeguato per creare applicazioni in cui si vogliono utilizzare un certo numero di robot poichè non è possibile una comunicazione diretta tra 2 Slave, occorrerebbe sempre avere la mediazione del Master.

Così come non vi sono possibilità a livello hardware di poter cambiare questo tipo di gestione delle connessioni, a livello software non vi sono

primitive che permettono scambio diretto di messaggi tra 2 Slave: la comunicazione è basata esclusivamente su una richiesta del Master e risposta del corrispondente Slave o viceversa. Questo non permette di avere una comunicazione universale ed immediata tra tutti i robot e risulta quindi difficile poter coordinarli per un lavoro cooperativo. Inoltre in questo modo il Master, rispetto agli altri robot, sarà il device che si caricherà di maggior lavoro e se non sarà sufficientemente veloce per esaudire tutte le richieste è facile pensare che potranno verificarsi inconvenienti e situazioni anomale.

2.3 Il protocollo di comunicazione

La comunicazione avviene tramite scambio di messaggi: un Master invia una richiesta ad uno Slave, specificandone la linea a cui è connesso (ossia i canali 1, 2 o 3) e quest'ultimo potrà rispondergli inviando a sua volta un messaggio con un contenuto, ma senza dover esplicitare il destinatario, poichè secondo le politiche di comunicazione deve essere necessariamente il Master sulla linea 0. È possibile inoltre mandare messaggi per impartire comandi diretti: in questo caso spesso il Master non si aspetterà un messaggio di ritorno.

L'interazione è stata progettata in tal modo ed ottimizzata partendo dalla supposizione che sia il Master ad iniziare la comunicazione inviando un messaggio di richiesta ad uno Slave.

Per mantenere momentaneamente i messaggi si fa uso di particolari buffer: le mailbox. Per ogni connessione ne vengono messe a disposizione 10: l'uso che ne fa il Master è un po' più complesso rispetto a quello dello Slave, perchè, oltre alla mailbox, esso deve specificare tramite il canale di connessione, di quale altro dispositivo appartiene. Dopo l'invio, il messaggio viene recapitato nella mailbox del robot corrispondente, che, non appena possibile, lo preleverà e ne leggerà il contenuto.

Dal punto di vista ingegneristico è molto importante il fatto di poter sfruttare mailbox multiple ed utilizzare ognuna di esse per contenere tipi di dati differenti in modo tale da lasciarli separati.

Supponendo che il Master invii immediatamente i messaggi e li riceva solo quando interroga gli Slave (cioè dopo aver eseguito il polling), si

presuppone che solo questi ultimi abbiano bisogno di code per tenere temporaneamente i messaggi.

I messaggi sono gestiti da un solo Thread perchè il canale trasmissivo ad onde radio è sequenziale: basta un solo Thread per tenerlo occupato.

Il modulo applicativo Bluetooth è basato su un protocollo non affidabile di alto livello, il Serial-Port Profile (SPP), che si appoggia su un protocollo di basso livello, l'RFCOMM, che invece garantisce l'integrità dei dati trasmessi.

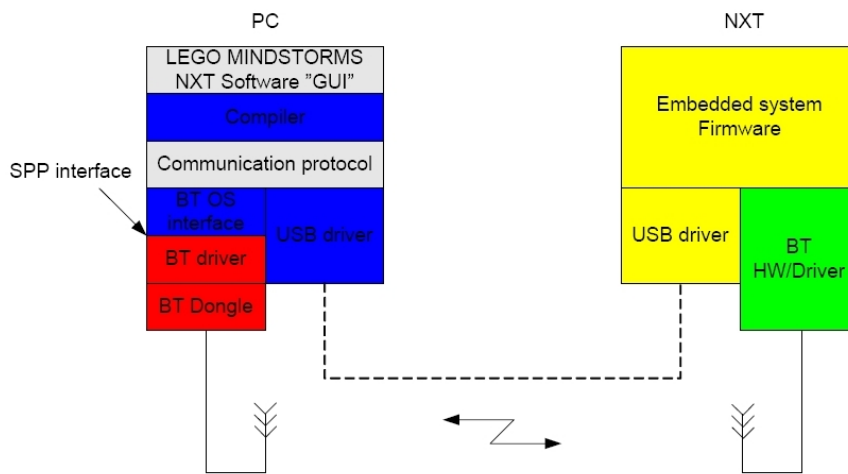


Figura 2.2: Diagramma di comunicazione

L'inaffidabilità dell'SPP è data dal fatto che per default il Master non richiede un ACK dell'avvenuta e corretta ricezione da parte dello Slave. Questo modello si presenta quindi molto diverso dai normali modelli adottati solitamente, che, al contrario, utilizzano un protocollo di alto livello affidabile che cerca di riparare ad errori di trasmissione del protocollo di più basso livello (come avviene nel modello TCP/IP).

2.3.1 Limiti del supporto Bluetooth nativo

Sono stati riscontrati numerosi problemi legati soprattutto all' hardware limitato di cui dispongono i robot.

Innanzitutto bisogna ricordare che questi robot possiedono una memoria molto esigua: infatti ogni mailbox riesce a contenere un massimo di soli 5 messaggi. Se il robot è troppo lento per processarli tutti e liberare spazio, a fronte di nuove ricezioni, se la mailbox dovesse risultare piena, verranno sovrascritti i messaggi più vecchi che quindi andranno perduti. Difatti mentre il protocollo RFCOMM garantisce una correttezza dei dati trasmessi, l'SPP non fornisce la certezza che essi vengano letti dal dispositivo. È possibile che avvenga una perdita di messaggi per differenti cause: il programma sullo Slave potrebbe essere troppo lento per leggere continui messaggi da parte del Master (la mailbox di input dello Slave va in overflow), altrimenti potrebbe verificarsi che il programma sul Master non sia abbastanza veloce nell'interrogare uno Slave che gli invia continuamente messaggi (overflow della mailbox di output dello Slave) o che lo Slave mandi più di 5 messaggi in fila senza interruzioni (in questo caso verranno tenuti solo gli ultimi). Altri motivi sono legati alla comunicazione: se non vi è nessun programma in esecuzione sullo Slave (i messaggi Bluetooth vengono ignorati), o se il programma non è comunque ancora pronto o, infine, se si interrompe il programma sullo Slave subito dopo aver mandato dei messaggi (la mailbox viene subito cancellata ed il Master non ha il tempo di fare il polling).

Un altro problema dovuto alla tecnologia adottata è il lungo tempo impiegato per effettuare lo switch da modalità di ricezione a quella di trasferimento e viceversa: questo è il motivo principale per cui viene adottato un protocollo inaffidabile: la ritrasmissione di un ACK di conferma dopo aver ricevuto un messaggio comporterebbe un passaggio immediato dello Slave da modalità ricezione a trasmissione e questo implicherebbe una lenta reattività. Inoltre l'invio di messaggi senza dati utili, cioè privi di significato come un ACK, porterebbe solo ad un'ulteriore perdita di tempo. Il protocollo Bluetooth, infatti, è poco performante ed estremamente lento: la trasmissione via onde radio è abbastanza veloce, tuttavia non lo è la comunicazione dalla CPU dell'NXT al modulo Bluetooth (utilizza una porta UART).

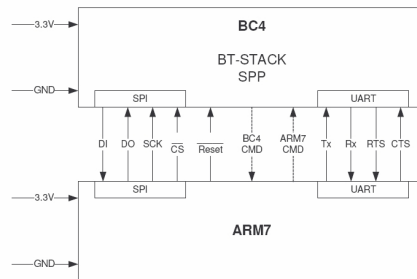


Figura 2.3: *Interfaccia Hardware tra ARM7 (processore) e il chip BlueCore*

Inoltre i messaggi contengono anche parti del protocollo NXT e non solo dati del programma (il Master invia 7 byte in più), e le risposte degli Slave sono sempre lunghe 66 byte per i dati, perdendo quindi molta efficienza se si tratta di inviare messaggi corti. Questo porta ad una sovrabbondanza di dati che devono essere trasmessi con un mezzo di comunicazione lento.

Un protocollo affidabile quindi porta spesso ad una perdita di messaggi, mentre uno che ne garantisce la ricezione è inevitabilmente affetto da lentezza.

L'approccio delle mailbox multiple è ottimo dal punto di vista ingegneristico tuttavia per il Master, sebbene non vi siano differenze dall'inviare un messaggio ad una singola, ricevere su più mailbox significa fare il polling separatamente per ognuna di quelle esistenti, comprese quelle che risultano vuote, ottenendo una bassa performance a causa del tempo perso.

Si possono usare anche più Thread sul Master per fare il polling, ma bisogna considerare che il canale radio ammette un solo Thread per volta, quindi per questa ragione non risulta molto efficiente utilizzare più Thread, perchè se questi dovessero interrogare la mailbox per messaggi il canale radio potrebbe rimanere occupato per molto tempo.

Inoltre la tecnologia Bluetooth offre grandi vantaggi ma non è possibile affermare che sia del tutto performante: in primo luogo la trasmissione è molto più lenta e inaffidabile, rispetto a quella via cavo.

Concludendo, adottando la politica di non inviare ACK, si ha un protocollo non sicuro che si appoggia su una tecnologia hardware anch'essa che non può fornire garanzie o che comunque è molto limitata: nel

complesso, il tutto sembra risultare molto inaffidabile e offrire scarse prestazioni.

2.4 Soluzioni per migliorare il protocollo

Dopo l'analisi del protocollo di base, in cui sono stati evidenziati i principali difetti, si può pensare a quali meccanismi sia meglio adottare per rendere innanzitutto il protocollo più agevole all'utente.

Si dovrebbe eliminare la distinzione tra Master e Slave, per avere tutti i robot sullo stesso livello: in questo modo essi potrebbero comunicare tra loro.

Per un sistema formato da 2 soli robot, non conviene attuare un cambiamento nella politica di gestione della comunicazione perchè tra essi esiste già un collegamento. Se, invece, si vogliono fare interagire 3 robot occorrerebbe creare un meccanismo che permetta ai robot di avere un 'link virtuale' che riesca a farli comunicare tra loro. In questo ultimo caso i robot devono essere tutti alla pari, quindi si possono utilizzare i 3 Slave, mentre il Master, che a livello hardware ha un comportamento molto differente, si preoccuperà di creare questo collegamento fittizio tra i 3 robot. Tuttavia questo indubbiamente presuppone che al massimo il sistema sia formato da 3 robot, il quarto (il Master), anche se presente, avrà solo il compito di permettere l'interazione fra quelli restanti.

Inoltre si potrebbe fare in modo di nascondere all'utente aspetti di più basso livello, come il numero della connessione e delle mailbox: lasciare che se ne occupi automaticamente il sistema sarebbe un aspetto che agevolerebbe all'utente le operazioni di comunicazione.

Tutto ciò presuppone l'introduzione e quindi l'implementazione di nuove primitive che permettano di avere un più alto livello di astrazione rispetto a quelle già fornite e che sfruttino un nuovo meccanismo di gestione della comunicazione.

2.5 Altre considerazioni

Dal punto di vista dell'affidabilità in questa tesi non si vogliono introdurre miglioramenti software ma comunque si possono fare già delle

considerazioni sulla base dell'analisi precedente nel caso in futuro si voglia toccare anche questo aspetto.

Innanzitutto è possibile adottare due differenti politiche quando si ha un protocollo inaffidabile: si può lasciare che i messaggi vadano perduti o si può intervenire bloccando anticipatamente il mittente quando la coda della mailbox risulta piena, ovviamente una soluzione esclude l'altra.

Si può utilizzare, inoltre, un altro meccanismo bloccante per non perdere i messaggi: mandare un solo messaggio per volta e inviare il successivo solo quando la mailbox risulta essere vuota.

Per fare in modo che non vi siano problemi alla connessione è bene inoltre iniziare la comunicazione con un messaggio dello Slave e chiuderla con uno del Master: in questo modo si è certi che all'apertura lo Slave sia operativo e che alla chiusura il Master abbia ricevuto tutti i messaggi dallo Slave.

Per non essere limitati dalla scarsa disponibilità di memoria si può utilizzare il PC per avere lunghe code, sicuramente i messaggi meno recenti non andranno sovrascritti, tuttavia dato che il protocollo di base è ottimizzato per avere le code principalmente sugli Slave, bisognerà che il PC abbia questo ruolo, anche se la funzione di Master sarebbe quella più adatta perchè in un sistema centralizzato è meglio che questo compito sia affidato alla macchina più potente e con maggiore velocità di calcolo.

La perdita di un messaggio non è sempre un aspetto negativo: se quest'ultimo diventa troppo vecchio è inutile prenderlo ancora in considerazione (per esempio per eseguire un comando), sul PC che può avere code molto lunghe ci potrebbe essere anche la possibilità di catalogarli per poterli eliminare automaticamente se troppo vecchi. Se tuttavia si utilizza invece il PC come Master, questo può inviare un messaggio ad uno Slave NXT specificando nell'header del messaggio se vuole una conferma tramite un codice di successo/errore dell'operazione, a questo punto lo Slave manderà una risposta quando è capace di mettere in coda il messaggio senza che essa vada in overflow.

Non vi è convenienza nell'usare mailbox multiple ed è migliore fare il polling sequenzialmente, utilizzando un solo Thread, per i motivi descritti nel paragrafo precedente.

Infine non è preferibile l'uso di un protocollo rispetto ad un altro,

poichè dipende dal tipo di programma in cui esso verrà applicato: se si preferisce la velocità a discapito dell'affidabilità allora basta solo migliorare il protocollo di base, se invece occorre avere maggiore affidabilità bisogna introdurre meccanismi più complessi, questo però porterà inevitabilmente ad una lentezza nella trasmissione ed elaborazione dei dati scambiati.

Capitolo 3

Modelli di comunicazione di alto livello

In questo capitolo verranno analizzati diversi tipi di modelli di comunicazione tra due o più entità in modo tale da consentire l'interazione.

3.1 Introduzione

Si prenderanno in esame alcuni tipi di modelli di comunicazione tra i più diffusi per stabilire successivamente quale sia il più adatto alle nostre esigenze. I modelli vengono definiti a livello teorico, perciò non dipendono dal tipo di hardware a cui verranno applicati o dal tipo di linguaggio di programmazione: si tratta di capire solo quale sia il migliore da seguire. La scelta finale, tuttavia, sarà dettata comunque dallo scopo che si vuol raggiungere e dalle risorse messe a disposizione: tale modello infatti dovrà poi essere il più adeguato alle componenti fisiche, alle potenzialità del software per poterlo poi implementare ed al grado di performance necessaria.

Un modello di comunicazione è composto da un medium di comunicazione, ossia un mezzo attraverso il quale sia possibile che avvenga un'interazione, e da alcune primitive che permettono le operazioni di comunicazione; se l'interazione è diretta, il medium è solo idealizzato. Spesso bisogna aggiungere anche alcuni meccanismi che regolamentino gli accessi se occorre gestire risorse condivise, situazione tipica dei sistemi concorrenti. Questi meccanismi possono essere semafori o moni-

tor, che bloccano l'accesso alla risorsa nel caso questa sia già occupata o comunque utilizzata da un'altra entità.

I modelli possono venire applicati a qualsiasi sistema: Internet, una rete locale, parti di un'applicazione stand alone, o, nel nostro caso, a dei robot. Di seguito verranno illustrati alcuni principali modelli di comunicazione, soffermandoci sul *Message Passing* e sullo *spazio di tuple*. Ogni modello non ha un unico modo per essere realizzato, si possono prendere in considerazione diverse varianti, dettate da fattori e vincoli, che sicuramente incidono sulle prestazioni finali.

3.2 Modelli di comunicazione diretta

Due entità comunicano tra di loro direttamente quando non fanno ricorso ad alcuno strumento che funge da intermediario, o almeno si astrae dall'idea di avere un medium di comunicazione.

La comunicazione è veloce, immediata ed avviene tramite un canale trasmissivo.

3.2.1 Scambio di messaggi diretto

La comunicazione attraverso scambio di messaggi (*Message Passing*) è uno dei più diffusi modelli di interazione esistenti. Abbastanza semplice da realizzare rispetto ad altri modelli, ma comunque molto efficace, è utilizzato in sistemi complessi, come i sistemi operativi, e anche per esempio nella maggior parte delle tecnologie object-oriented distribuite, come i Web Services; questo tipo di comunicazione è quella tipica che avviene per esempio tra i nodi della rete Internet.

Uno scambio di messaggi diretto implica che sia realizzato un unico collegamento virtuale che sia valido solo per una coppia di agenti: essi comunicano tra di loro tramite messaggi specificando in maniera esplicita il destinatario (o il mittente in caso di ricezione).

Il Message Passing si basa sull'utilizzo di due richieste (primitive): la *send* e la *receive*, realizzabili anche con diverse varianti.

Figura 3.1: *Message Passing diretto*

Nella comunicazione diretta il modello di una *send* è tipicamente:

send(B, Messaggio);

mentre quello della *receive*:

receive(A, Messaggio);

Se consideriamo *A* come il mittente del messaggio e *B* il destinatario; questo tipo di comunicazione è simmetrica. Si può realizzare tale modello in forma asimmetrica, molto più flessibile:

send(B, Messaggio);

receive(id, Messaggio);

In tal modo il destinatario può ricevere un messaggio da qualsiasi altra entità. L'*id* viene riconosciuto ed istanziato dinamicamente come mittente.

Si può decidere inoltre se realizzare lo scambio di messaggi in maniera bloccante (sincrono) o non bloccante (asincrono). Una *send* è bloccante se il mittente interrompe qualsiasi operazione finché il destinatario non riceve il messaggio, mentre con una *receive* bloccante il destinatario si ferma fino a quando riceve un messaggio. *Send* e *receive* asincrone, invece, permettono a chi le invoca di proseguire senza interrompersi; il destinatario che esegue una *receive* asincrona deve effettuare un controllo periodico per accorgersi della presenza di un nuovo messaggio.

3.3 Modelli di comunicazione indiretta

La comunicazione indiretta avviene grazie ad un medium di comunicazione che funge da tramite, in genere uno spazio esterno, comune alle due unità interagenti, in cui possa avvenire lo scambio di informazioni.

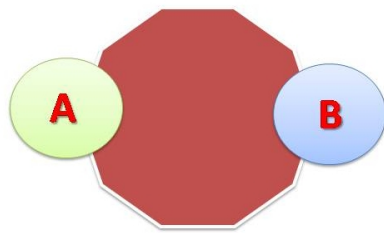


Figura 3.2: *Medium di comunicazione/coordinazione*

3.3.1 Scambio di messaggi indiretto

Il *Message Passing* indiretto si può realizzare grazie a mailbox (chiamate anche messagebox) o porte come medium di coordinazione. I messaggi vengono inseriti dal mittente nella mailbox, senza giungere direttamente al destinatario, che potrà invece prelevarli in un secondo momento. Il meccanismo è molto semplice: grazie ad una *send*, il mittente recapita il messaggio in una mailbox condivisa con il destinatario, il quale farà una *receive* da quest'ultima per poterlo leggere.

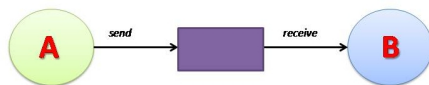


Figura 3.3: *Message Passing indiretto*

Nella comunicazione indiretta il modello di una send è tipicamente:
send(MailBox, Messaggio);

mentre quello della receive:
receive(MailBox, Messaggio);

Questo modello di primitive tuttavia è opportuno usarlo solo nel caso vi sia una mailbox per ogni singola coppia di entità che vogliono comunicare: se vi sono più mittenti o destinatari che utilizzano la stessa mailbox occorre fare in modo di specificare anche a chi è indirizzato il messaggio o da chi proviene, tuttavia bisogna tener conto del fine della comunicazione: a volte non è detto che sia utile riportare mittente e destinatario anche con sistemi altamente concorrenti (per esempio per messaggi broadcast, o cmq diretti a qualsiasi destinatario).

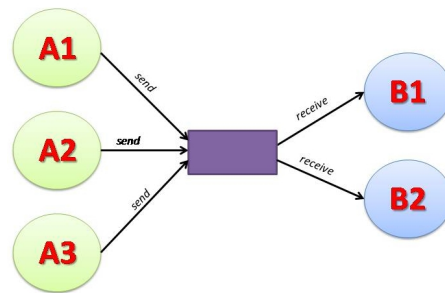


Figura 3.4: Message Passing indiretto con mailbox condivisa

Avendo come intermediario una mailbox, la versione bloccante può essere realizzata in diversi modi: con una send completamente sincrona, cioè il mittente attende che il ricevente abbia prelevato il messaggio prima di poter proseguire, o altrimenti si può limitare il bloccaggio al solo inserimento nella mailbox. Quest'ultima, infatti, avrà certamente una capacità massima, perciò bloccherà nuovi messaggi in arrivo nel caso essa sia piena, fino a quando non verranno liberati nuovi posti; il mittente, quindi, rimane bloccato in attesa di mettere il proprio messaggio nella mailbox, tuttavia non deve aspettare che il destinatario lo abbia prelevato.

3.3.2 Modelli basati su spazi

Immaginiamo di avere virtualmente una lavagna su cui tutti possono attaccare e lasciare fogliettini per qualcuno e che questi ultimi possano poi essere letti o staccati da chiunque ne sia interessato. Tale lavagna sarà il medium di coordinazione che chiameremo *space* ed i fogliettini le informazioni che esso può contenere. Ovviamente lo *space* è un mezzo di comunicazione totalmente condiviso, su cui interagiscono agenti esterni, e bisognerà adottare una politica adeguata per controllare gli accessi, tipica dei sistemi concorrenti.

Introducendo i modelli *space based* si fa spesso riferimento allo *spazio di tuple*. Questo tipo di approccio sfrutta come *space* memoria di tipo associativo: le tuple, che sono ‘raccolte’ di informazioni (ossia informazioni suddivise in più campi), infatti vengono caricate e lette in tale spazio, dopo essere state identificate tramite matching su una chiave, piuttosto che utilizzare un meccanismo tradizionale di indirizzamento.

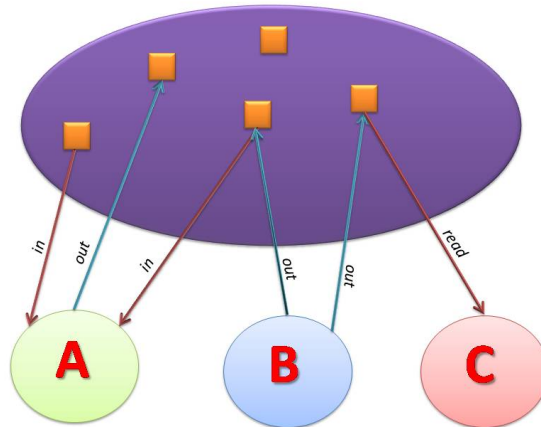


Figura 3.5: Spazio di tuple

Tale genere di modello, altamente concorrente, prevede tipicamente alcune particolari primitive che permettono di eseguire le operazioni necessarie:

Inserimento di una tupla:

out(s);

Prelievo di una tupla:

in(s);

Lettura di una tupla, senza eliminarla:

read(s);

La *in* e la *read* sono bloccanti ed in generale anche la *out*.

È un modello di comunicazione che ha un alto grado di astrazione, ma è più difficile da realizzare per la sua complessità e per la gestione degli accessi, inoltre occorrono anche particolari risorse, e non sempre quelle disponibili lo permettono.

3.4 Confronto fra le due tipologie di comunicazione

Nella comunicazione diretta, il link fra 2 entità viene stabilito automaticamente e comunque è dedicato ad una sola coppia; può essere unidirezionale o bidirezionale. Non necessita di particolari attenzioni per quel che riguarda il controllo dell'accesso alle informazioni da parte di terzi, o comunque di entità non desiderate, tuttavia è di basso livello dal punto di vista ingegneristico: la sua scarsa modularità non permette di essere riutilizzato, occorre, infatti, riorganizzare e cambiare il codice nel caso si volessero stabilire link diversi per fare comunicare entità differenti. Inoltre occorre realizzare un apposito link per ogni coppia, quindi si avranno moltissimi collegamenti point to point da creare e gestire.

Avendo, invece, una comunicazione indiretta, il link è stabilito al momento in cui le entità condividono il medium di comunicazione; quest'ultimo può essere associato a più entità ed ogni coppia può

avere più link, per esempio scambiando messaggi tramite più mailbox. Anche in questo caso i collegamenti possono essere unidirezionali o bidirezionali. Il problema fondamentale di questo modello riguarda l'accesso al medium di comunicazione, ossia possono sorgere problemi di sincronizzazione: ovviamente con questo genere di interazione si ha un sistema concorrente dato che più entità condividono la stessa risorsa, occorre perciò adottare meccanismi adeguati per non incontrare errori e fare in modo che le entità abbiano accesso ordinatamente e mutuamente alle informazioni.

Con una comunicazione indiretta, quindi, il sistema è più flessibile e modulare, tuttavia in generale risulta assai più complesso da progettare e gestire.

3.5 Considerazioni

Il modello basato su spazi possiede indubbiamente numerosi vantaggi: ogni entità può inserire e prelevare le informazioni desiderate in qualsiasi momento, inoltre le tuple permangono nello spazio. Occorre, tuttavia, avere uno spazio di memoria abbastanza capiente da poter far fronte a tutti gli inserimenti, ma soprattutto l'entità che ospita e gestisce lo space dovrà avere anche un'elevata velocità di calcolo per permettere che vengano eseguiti più task contemporaneamente e soddisfare tutte le richieste.

Il più comune e diffuso approccio rimane comunque quello a scambio di messaggi perchè è indubbiamente più immediato da realizzare.

Se si volessero considerare anche le prestazioni, si potrebbe stabilire un compromesso tra performance ed affidabilità concentrandosi nel caso indiretto su una send parzialmente bloccante, cioè che riceve conferma solo dell'inserimento di un messaggio nella mailbox, ma non che esso venga realmente ricevuto: un sistema senza alcun controllo, ovviamente non è affidabile, tuttavia, adottando una politica troppo rigida e pesante, invece, si rischia di sprecare inutilmente troppo tempo e risorse (a causa degli ACK, messaggi privi di contenuto).

Capitolo 4

API per la comunicazione diretta

In questo capitolo viene fornita una soluzione software di alto livello basata sui modelli descritti nel capitolo precedente per rendere più agevole la comunicazione. In particolare si cercherà di permettere ai robot di comunicare in maniera diretta.

4.1 Obiettivo

I LEGO Mindstorms come descritto già in precedenza interagiscono tra di loro con una forma di Message Passing indiretta: si avvale infatti di scambio di messaggi tra mailbox. Delle 10 mailbox messe a disposizione per connessione tra Master e Slave in grado di accogliere i messaggi, in generale ne bastano solo 2, una per i messaggi in entrata ed una per quelli in uscita.

Il canale trasmissivo è dato da un collegamento Bluetooth come illustrato nei capitoli precedenti; i Brick NXT, sono collegati tra loro, tramite canali radio a cui sono associate le mailbox. A livello logico possiamo supporre che ogni Slave possieda una mailbox in cui può riporre, tramite una Send, un messaggio per il Master e che quest'ultimo esegua il polling tramite una Receive sulla stessa mailbox per poterlo prelevare. Un'altra mailbox servirà per accogliere messaggi o risposte da parte del Master; vi è una continua interazione fra le mailbox e Brick NXT.

Ciascuna unità esterna può comunicare tramite tale sistema grazie alla mailbox associata a quest'ultimo. Ovviamente il protocollo NXT permette di associare anche più mailbox, o gestirle in maniera differente, ma comunemente per ottenere migliori risultati nella comunicazione è conveniente strutturarlo in tal modo. Le primitive messe a disposizione dal linguaggio NXC si basano sulle connessioni e su mailbox sulle quali viene eseguito il polling. Uno Slave, potendo comunicare soltanto con il Master, devono far seguire ai propri messaggi un 'percorso' non immediato per poter interagire con un altro Slave, inoltre deve fare riferimento sempre e solo al Master.

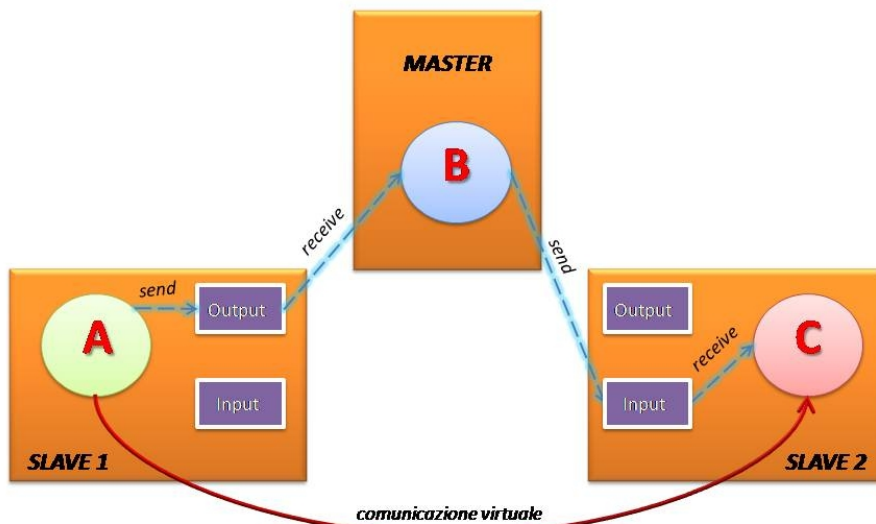


Figura 4.1: *Comunicazione tra 2 Slave*

Il Master funge da intermediario tra i 2 sistemi che vogliono comunicare.

Sarebbe allora opportuno realizzare un meccanismo che a livello logico possa mettere in comunicazione 2 Slave tra loro, senza dover preoccuparsi di fare interagire il Master. Questo ovviamente sarà quello che dovrà apparire ad alto livello, perchè comunque a livello hardware (che rimane molto limitato) il protocollo non può funzionare diversamente

da quello esistente; quindi d'ora in avanti si cercherà di pensare di non avere più un sistema formato da Master e Slave, ma considerare di avere semplici robot che interagiscono tra di loro senza gerarchie, grazie all'ausilio di un mediatore che comunque rimane estraneo alla comunicazione e che l'utente non deve considerare durante la programmazione dei restanti robot. Inoltre è preferibile una comunicazione che non faccia riferimento ad alcun medium di comunicazione, quindi si cercherà di passare dalla forma indiretta del Message Passing a quella diretta, vista nel capitolo precedente, eliminando la gestione delle mailbox ed in tal modo agevolando all'utente la gestione dell'interazione. Questo sarà permesso dalla realizzazione di una nuova libreria, che per il momento si baserà su un protocollo non affidabile, perchè è il meno complesso.

4.2 Le nuove API per la comunicazione diretta

Si è deciso quindi di implementare delle nuove API che permettano uno scambio diretto di messaggi e che andranno a sostituire le vecchie primitive di più basso livello fornite di base dal linguaggio NXC. Esse saranno contenute nella libreria *NXTCommLib.nxc*.

Come detto in precedenza esse saranno ancora basate su un protocollo non affidabile, tuttavia esse danno la possibilità di avere una comunicazione molto più agevole.

In alcuni casi le primitive sono state realizzate in versioni differenti, come nel caso della *Receive*, in forma bloccante e non bloccante; la *Send* è per ora esclusivamente non bloccante.

Per poterle sfruttare occorre importare la libreria, includendola all'inizio del programma desiderato:

```
#include 'NXTCommLib.nxc'
```

Le nuove primitive che fornisce tale libreria sono:

```
void InitComm(byte id);
```

È utilizzata per l'inizializzazione di un robot. Occorre invocarla una sola volta all'inizio del programma per rendere possibile poi l'utilizzo delle restanti API della nuova libreria. Il robot si mette in attesa di un messaggio da parte del mediatore e poi si registrerà ad esso. Occorre specificare il numero di ID (da 0 a 9) con cui vuole essere identificato dagli altri dispositivi. È ad opera dell'utente fare attenzione a non utilizzare lo stesso ID per device diversi (ci sarebbe un problema di ambiguità). Bisogna anche considerare che lo 0 è riservato al mediatore, a meno di variazioni nel codice del programma in esecuzione su di esso.

```
void Send(byte id, string message);
```

È il modello di Send diretta, per inviare un messaggio ad un altro device. Specificando l'ID del destinatario ed il messaggio (una stringa di caratteri), quest'ultimo giungerà al device desiderato, salvo errori.

```
void SendToAll(string message);
```

È una variante della Send: permette di inviare un messaggio broadcast, ossia contemporaneamente a tutti i dispositivi che fanno parte del sistema, senza quindi specificare nessun ID particolare.

```
string Receive();
```

Con questa primitiva l'NXT riceve in maniera diretta tutti i messaggi destinati ad esso, provenienti da qualsiasi altro dispositivo. La funzione ritorna infatti il messaggio inviato da qualcuno tramite una Send. In maniera automatica, dopo ogni Receive, viene registrato il mittente nella variabile globale *LAST_SENDER*, che, se richiamata, permette quindi di conoscere il mittente dell'ultimo messaggio; viene sovrascritta con un nuovo valore dopo la successiva Receive andata a buon fine.

```
string BlockingReceive();
```

È la versione bloccante della *Receive*: il processo si mette in attesa e non prosegue con l'esecuzione delle successive istruzioni fino a quando non viene ricevuto un messaggio.

```
string ReceiveFromID(byte id);
```

È una variante della *Receive*: permette di ricevere messaggi esclusivamente dal mittente specificato nell'argomento. I messaggi restanti provenienti da altri NXT vengono scartati.

Utilizzando queste API è dunque possibile far comunicare 2 o più robot tramite scambio di messaggi come se fra essi vi fosse un link diretto.

4.3 Architettura della libreria

La libreria fornisce le API per i robot che si vogliono fare interagire, pertanto essa sarà inclusa nei programmi che dovranno essere eseguiti sui singoli Brick, tuttavia per permettere che essa funzioni occorre avere anche un mediatore che abbia attivo un programma implementato ad hoc che non fa parte della libreria, ma che è sempre valido per ogni tipo di applicazione che invece verrà messa in esecuzione sui singoli robot.

4.3.1 La struttura

Il sistema comprende un mediatore che ha il compito di ricevere ed instradare i messaggi degli altri 3 robot a lui collegati.

L'identificativo (ID) attribuito ad ogni robot per riconoscersi è importante perchè permette di astrarre dalla connessione fisica (linea 1, 2 o 3); per convenzione deve essere compreso tra 0 e 9, perchè la capacità di testo di un messaggio Bluetooth è assai limitata (58 byte), quindi si è deciso di sfruttare solamente un unico byte, il primo del messaggio, per far contenere questa informazione. Inizialmente, infatti, si era pensato di utilizzare un vero e proprio nome, ma questo significava essere

costretti ad utilizzare e gestire 2 mailbox in uscita per robot (una per il nome e la seconda per il messaggio) con conseguenti problemi indesiderati di sincronizzazione, di accesso e di ritardi troppo elevati (il mediatore doveva attendere di avere entrambi i messaggi, per poterlo poi spedire), quindi si è preferito avere informazioni sul messaggio e sull'utente nel testo di unico messaggio. Il messaggio quindi verrà inviato come stringa di caratteri, codificata dal mittente e decodificata dal mediatore, che poi assocerà l'ID alla vera connessione del robot ricevente. L'ID del mittente viene poi aggiunto nel primo byte della stringa instradata dal mediatore al destinatario; il messaggio poi viene decodificato dal robot ricevente, che quindi può saperne anche la provenienza.

Per lo scambio di messaggi si è deciso di attribuire mailbox diverse a seconda della linea di connessione, questo per evitare ambiguità: in questo modo il mediatore, in base alla mailbox in cui risiede il messaggio può sapere con certezza da chi proviene. È stato necessario poichè il mediatore può spedire un messaggio ad uno specifico robot, tuttavia può ricevere da una determinata mailbox senza dichiarare la linea di connessione ed in questo modo non può conoscere da chi proviene. Essendoci a disposizione 10 mailbox si è pensato di attribuire la 1 e la 5 per la connessione sulla linea 1, la 2 e la 6 per la linea 2, la 3 e la 7 per la 3: una mailbox serve per accogliere i messaggi in entrata per il robot, l'altra per gestire quelli in uscita.

Tutti i device hanno la possibilità di comunicare tra di loro, potendo ricevere ed inviare allo stesso tempo.

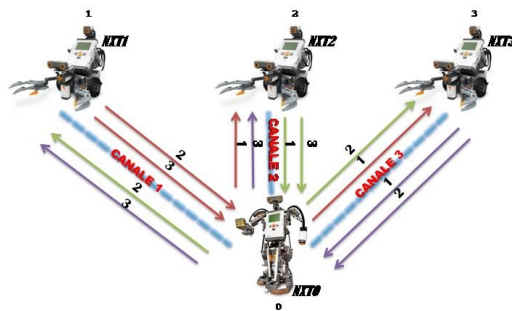


Figura 4.2: *Comunicazione bidirezionale tra 3 NXT*

Questo era uno schema logico semplificato poichè infatti bisogna ricordare che comunque anche tra robot e mediatore non vi è comunicazione diretta, poichè si fanno ricorso a mailbox.

Si è escluso di dare la possibilità al mediatore di fare parte del sistema ed essere attivo anch'esso nell'interazione: ha solo il compito di instradare i messaggi. Questo è per evitare tempi troppi lunghi o perdita di messaggi, dovuti ad un carico troppo pesante di lavoro affidato ad esso.

Per avere un codice flessibile, non prefissato e quindi che si adatta automaticamente di volta in volta a seconda delle circostanze, è stato realizzato un meccanismo di assegnamento dinamico. In principio l'assegnamento delle mailbox degli NXT era prefissato, in base alla linea in cui erano connessi: ogni robot non era in grado di riconoscere il numero del proprio canale comunicativo assegnato a livello hardware e per questo bisognava farglielo sapere, inserendolo tali informazioni nel codice a priori.

Il mediatore, come azione preliminare, controlla quali dispositivi sono connessi, successivamente manda un messaggio contenente i numeri delle mailbox da utilizzare (dipendenti dalla connessione), nella mailbox 4 sulla linea in cui è stato rilevato un NXT. In seguito, dopo essere stato in attesa di ricevere tale informazione, l'NXT risponde al mediatore con un messaggio contenente il proprio ID, che verrà quindi associato a quella linea.

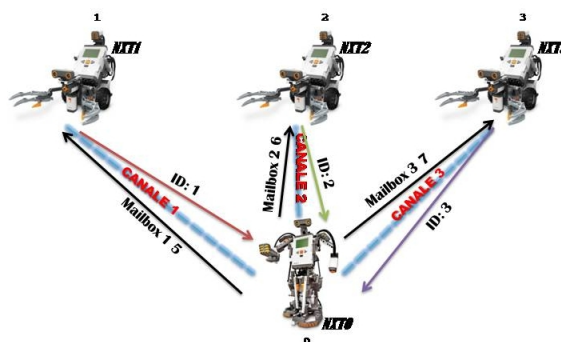


Figura 4.3: Inizializzazione per registrare i robot al mediatore

In questo modo l'ID dipende solo dal programma specifico eseguito su ogni robot e non dalla reale connessione Bluetooth. È indifferente scegliere di collegare un particolare NXT ad un canale o all'altro, si astrae da questo permettendo all'utente di non preoccuparsene.

La libreria mette a disposizione nuove API di alto livello con cui è possibile far comunicare direttamente a livello logico 2 robot tra di loro, senza curarsi del fatto che nella realtà, invece, a livello sia software che hardware il percorso per fare recapitare un messaggio è molto più lungo e complicato e che esiste a basso livello una distinzione di ruoli che non dà la possibilità di comunicazione diretta.

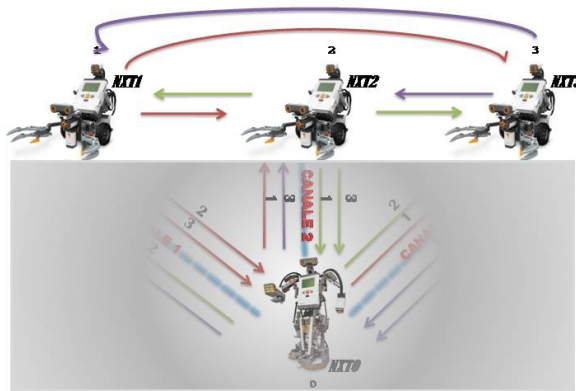


Figura 4.4: *Librerie di comunicazione*

4.3.2 Il mediatore

Il mediatore, quello che a basso livello è il Master NXT, ha una propria applicazione (*Mediator.nxc*) ed è usato solo come supporto alla comunicazione dei restanti robot. Come detto in precedenza non sarebbe molto performante decidere che anch'esso prenda parte attiva alla comunicazione, per problemi dovuti ad un carico troppo pesante di elaborazione.

In esso è richiamata *StartInitComm(byte id)*, una funzione di inizializzazione che permette di individuare e registrare i robot presenti (la corrispondente *InitComm(byte id)* di quest'ultimi); come argomento

accetta un numero che deve andare dallo 0 al 9: è il proprio identificativo. Di default il programma gli assegna l'ID 0, pertanto è opportuno non utilizzarlo per nominare uno dei rimanenti robot.

Ogni informazione riguardante un NXT (connessione, ID e mailbox utilizzate) viene poi memorizzata in una struttura dati chiamata *device*.

Il main dell'applicazione non fa altro che controllare periodicamente se da parte di un dispositivo collegato vi è un messaggio destinato a qualcuno: il mediatore esegue un lavoro di smistamento continuamente prima per i messaggi provenienti dalla connessione 1, poi per quelli della 2 ed infine per la 3, in maniera sequenziale.

Dopo aver rilevato un messaggio ed averlo decodificato, viene richiamata un'altra funzione che invece associa l'ID del destinatario alla linea effettiva del canale, quindi codifica il messaggio e lo inoltra al giusto destinatario, inserendo anche come informazione l'ID del mittente.

Il mediatore non ha bisogno della libreria poichè non partecipa attivamente alla comunicazione: è sufficiente il programma specifico che deve eseguire per far permettere la comunicazione.

4.4 Implementazione

Per l'implementazione e la compilazione è stato usato un pc con sistema operativo Windows XP Professional Service Pack 2.

Come detto precedentemente, il linguaggio scelto per la programmazione è l'NXC; per poter programmare e compilare sorgenti in tale linguaggio è necessario, come già accennato nel primo capitolo, l'IDE di sviluppo Bricx Command Center, non sviluppato dalla LEGO ma da terzi.

Il codice della libreria non è molto complesso e neanche lungo, tuttavia spesso sono stati necessari piccoli accorgimenti e test per ottenere a runtime i risultati sperati, inoltre essa ha richiesto che fosse sviluppato un secondo programma più complesso che rimanesse poi costantemente in esecuzione nel mediatore, e che non riporteremo qui.

```
#include "NXCDefs.h"
#define BT_CONN 0

byte MY_ID;
byte ID_0;

byte INBOX;
byte OUTBOX;

byte LAST_SENDER;

void InitComm(byte nxtID){
    string initRic;
    string cStr;
    string ID_OStr;
    string INBOXStr;
    string OUTBOXStr;

    MY_ID = nxtID;

    string cStr1 = NumToStr(0);
    cStr = StrCat("on line", cStr1);
    if (BluetoothStatus(0) != NO_ERR){

        TextOut(5,LCD_LINE1,"*Connect NXT",true);
        TextOut(5,LCD_LINE2, cStr,false);
        TextOut(60,LCD_LINE3, "from",false);
        TextOut(5,LCD_LINE4, "master NXT menu",false);

        Wait(3000);
        Stop(true);

    }else{

        until(ReceiveRemoteString(9, true, initRic) == NO_ERR);

        ID_OStr = SubStr(initRic,0,1);
        ID_0 = StrToNum(ID_OStr);
        INBOXStr = SubStr(initRic,1,1);
        OUTBOXStr = SubStr(initRic,2,1);
        string connStr = SubStr(initRic,3,1);
        INBOX = StrToNum(INBOXStr);
        OUTBOX = StrToNum(OUTBOXStr);
        string idStr = NumToStr(MY_ID);
        string b = StrCat(connStr, idStr);

        Wait(200);
        until(BluetoothStatus(0) == NO_ERR);
        SendRemoteString(0, OUTBOX, b);
    }
}
```

```
void Send(byte conn, string mess){

    string stringTot;
    string connection = NumToStr(conn);
    stringTot = StrCat(connection, mess);

    until(BluetoothStatus(0) == NO_ERR);
    SendRemoteString(BT_CONN, OUTBOX, stringTot);
}

void SendToAll(string mess){

    string stringTot;
    stringTot = StrCat("A", mess);

    until(BluetoothStatus(0) == NO_ERR);
    SendRemoteString(BT_CONN, OUTBOX, stringTot);
}

string Receive(){

    string message;
    if(ReceiveRemoteString(INBOX, true, message) == NO_ERR){

        int len = StrLen(message)-1;
        string sender = SubStr(message, 0, 1);
        string msg = SubStr(message, 1, len);

        LAST_SENDER = StrToNum(sender);
        return msg;
    }
}

string BlockingReceive(){

    string message;
    until(ReceiveRemoteString(INBOX, true, message) != STAT_MSG_EMPTY_MAILBOX);

    int len = StrLen(message)-1;
    string sender = SubStr(message, 0, 1);
    string msg = SubStr(message, 1, len);

    LAST_SENDER = StrToNum(sender);
    return msg;
}

string ReceiveFromID(byte sendID){

    string msg = "";

    string message = Receive();
    if(LAST_SENDER == sendID){
        msg = message;
    }
    return msg;
}
```

4.5 Deployment

Verranno di seguito elencate alcune procedure ed alcuni aspetti fondamentali di carattere pratico utili per essere operativi: creare, modificare (quindi programmare in NXC) o utilizzare la libreria spiegando anche le fasi di routine per il setup dei robot.

4.5.1 Utilizzo del software

Nei robot è stato mantenuto il firmware standard fornito dalla LEGO (aggiornato alla versione 1.05); esso permette di poter eseguire anche programmi scritti in NXC, perchè al momento della compilazione vengono tradotti in NXT bytecode.

Per convenzione si sono usati nomi ben precisi per identificare i diversi robot: *NXT0*, *NXT1*, *NXT2* e *NXT3*. Per rinominare i Brick NXT si è fatto ricorso al software fornito dalla casa produttrice LEGO (con cui è possibile programmare in NXT-G), con una connessione USB del Brick NXT al pc.

Per scrivere programmi in NXC, invece, come accennato già nel primo capitolo, si è usato Bricx Command Center.

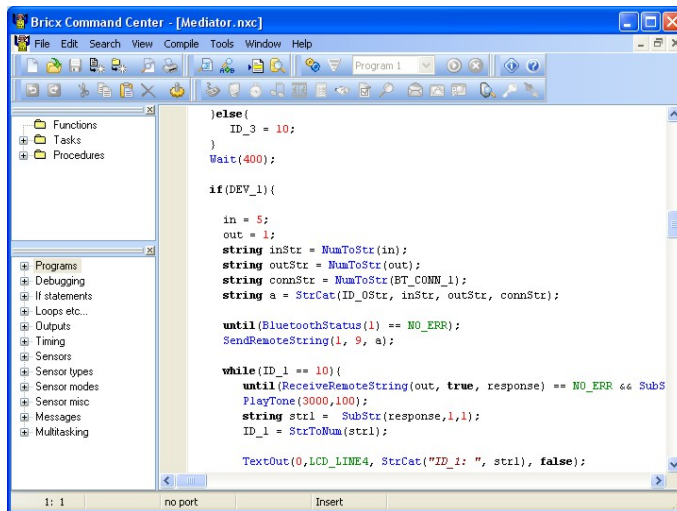


Figura 4.5: *Bricx Command Center*

Poichè tale IDE è adatto per la compilazione per diversi tipi sia di Brick che di linguaggi (anche per l'RCX o per lo Scout, meno recenti), bisogna selezionare l'NXT come tipo di prodotto e lasciare selezionato il firmware standard, inoltre occorre assicurarsi nelle opzioni di avere selezionato l'NXC come linguaggio.

Con BricxCC, una volta compilato il programma senza errori basta trasferirlo tramite l'apposito comando al Brick NXT, sempre che esso sia stato rilevato e connesso al computer.

Per ora ci si è limitati ad eseguire tale operazione grazie al cavo USB, ma è possibile un trasferimento anche via Bluetooth: quest'ultima soluzione è indubbiamente più comoda (non occorre scollegare e inserire il cavo continuamente), ma comporta un download più lento, inoltre sono stati riscontrati inizialmente alcuni problemi di driver che non hanno permesso di poterla eseguire, quindi, si è pensato di risolvere tale problemi solamente se il collegamento Bluetooth dovesse essere strettamente (per esempio per un'applicazione in cui si necessita di fare interagire anche il pc).

Come IDE di sviluppo, BricxCC è facile da utilizzare, tuttavia pecca di alcuni bug: per esempio non sempre il parser agisce correttamente. A volte, infatti, è capitato che non accettasse funzioni o variabili scritte in maniera esatta, mentre altre volte non rilevava errori in primitive in cui c'era sovrabbondanza di argomenti: è successo, ad esempio, che per un errore di implementazione, nella *ReceiveRemoteString* fosse stato aggiunto come primo parametro anche la linea di connessione, quando questo particolare tipo di funzione non la richiede; la interpretava come il numero di mailbox, il quale, invece, era inserito come secondo argomento. Le conseguenze si vedevano solo a run-time senza riuscire a capire se vi era un errore di scrittura, come infatti era. Inoltre in presenza di errori, anche se rilevati, spesso l'IDE non è chiaro nell'informare sull'origine e sulla sua posizione all'interno dell'intero codice.

Bisogna sapere anche che il programma esegue la compilazione in maniera sequenziale (cioè in ordine di come è stato implementato il codice): se si fa riferimento ad una funzione o una variabile dichiarata successivamente procedendo a leggere il codice in ordine, così come è stato scritto, anche se a livello logico il programma segue un flusso

corretto, in cui la dichiarazione precede l'uso di tale variabile, la considererà inesistente. Per questo motivo, il main, in genere è sempre posto come ultimo task, perchè utilizza tutte le altre funzioni. I programmi compilati vengono salvati sul Brick NXT come file di estensione *rxex*.

4.5.2 Operazioni preliminari

Prendiamo in considerazione per ora un sistema formato da 4 robot: per le particolarità di interazione dei robot Mindstorms discusse nei capitoli precedenti, a basso livello la rete sarà composta da un Master e tre Slave.

Innanzitutto, per poter utilizzare la libreria occorre realizzare fisicamente la rete ad onde radio, ossia predisporre l'hardware: per prima cosa dal menù *Bluetooth* di ogni robot bisogna rendere attivo tale funzionalità, impostandola su *on*. Una volta scelto il robot che avrà il ruolo di Master, sempre dal menù *Bluetooth* di quest'ultimo occorre fare una ricerca (*Search*) dei dispositivi con questa tecnologia che riesce a rilevare entro il proprio raggio (al massimo circa 10 metri). A questo punto compariranno fra i vari device anche i 3 Brick NXT a cui abbiamo attivato la ricezione Bluetooth: selezionandone uno, verrà successivamente chiesto a quale canale collegarlo fra quelli ancora disponibili (è probabile che venga chiesto anche il codice di sicurezza da entrambe le parti, che in genere è *1234*). Ricordando infine che il Master mette a disposizione 3 canali, attribuiamo ogni linea ad un robot differente.

È stato riscontrato che occorre ripetere questa operazione ogni volta che si è già fatto eseguire un'applicazione e la si vuole ripetere; è necessario per resettare il tutto, spegnere i Brick, e poi rifare le operazioni precedenti per ricreare i collegamenti, altrimenti l'applicazione molto probabilmente non risponderà.

Una volta eseguita questa operazione preliminare che permette di realizzare la rete, si potranno usare tranquillamente le API NXC per la comunicazione Bluetooth (ovviamente anche le nuove API appena realizzate).

Infine per far eseguire un'applicazione cooperativa che sfrutta le librerie è necessario azionare innanzitutto gli Slave poichè essi riman-

gono in attesa e come ultimo attivare il programma sul Master che darà il via alla comunicazione.

4.6 Collaudo

Per raggiungere il risultato finale sono state eseguite numerose prove, non solo per testare il parziale comportamento delle librerie in fase di sviluppo, ma anche per rendersi conto delle potenzialità e dei limiti dell'hardware di base.

4.6.1 La libreria

La libreria è stata realizzata per passi gradualmente, iniziando ovviamente da semplici punti, fino ad aggiungere ogni volta nuove funzionalità.

Si è partiti con il semplice presupposto di riuscire a far deviare dal mediatore un messaggio da un robot ad un altro, utilizzando le API NXC per la comunicazione Bluetooth con connessioni già prestabilite e non modificabili; come primi esperimenti è risultato più semplice utilizzare solo 2 robot (con il supporto ovviamente del mediatore).

Per testare si è pensato di far comparire sui vari LCD dei Brick i messaggi scambiati (ricevuti e inviati), o comunque informazioni sulla comunicazione. Per essere sicuri che il messaggio fosse inviato da una parte e ricevuto dall'altra, la *Send* e la *Receive* erano cicliche, cioè si ripetevano continuamente fino all'interruzione dell'esecuzione del programma. I messaggi giungevano comunque a distanza di tempo abbastanza considerevole a causa di numerose *Wait* usate soprattutto per far rimanere visualizzati a video i risultati.

Come secondo passo si è proceduto con attribuire un nome ad ogni robot per poter quindi fare recapitare un messaggio specificando il destinatario. Come descritto precedentemente, questa soluzione presupponeva 2 mailbox di output per il mittente: quest'ultimo eseguiva 2 *Send* ognuna indirizzata a mailbox diverse ed il mediatore da una prelevava il nome e dall'altra il messaggio; nel codice del mediatore era già associato il nome del robot alla linea a cui quest'ultimo era connesso. Ovviamente non era detto che in un ciclo un prelievo andasse a buon fine, quindi il mediatore doveva aspettare loop successivi per

avere entrambe le informazioni prima di inviarlo: questo aveva ripercussioni sui tempi di invio che risultavano molti lunghi, e comunque si è preferito abbandonare un'architettura di questo genere sebbene dai test sembrasse funzionare correttamente, pensando poi agli sviluppi futuri ed ai problemi che si potevano avere con un sistema più complesso.

Successivamente si è considerato più opportuno utilizzare la soluzione con l'ID, facendo comparire sullo schermo anche il destinatario, ed in seguito il mittente.

Si era pensato di inviare il messaggio sotto forma di array di byte o addirittura di *struct* ma i test riportavano sempre errore nelle Send e nelle Receive, allora alla fine si è deciso di lasciarlo come stringa.

Dopo questo test si è aggiunto di dare all'NXT ricevente l'opportunità di poter ottenere messaggi contemporaneamente da 2 robot, questo per provare che non vi fossero interferenze di sincronizzazione. In questa fase non vi sono stati particolari problemi.

Nell'ultimo passo ogni robot aveva abilitati sia il task per la ricezione che quello per l'invio, attivi in maniera ciclica: un NXT spediva un messaggio ad ognuno degli altri 2 robot e nel frattempo era in attesa di ricevere. I risultati venivano visualizzati mano a mano sul display (contenuto del messaggio, lunghezza, destinazione in caso di Send, provenienza in caso di Receive). Per avere questo sistema si sono fatti ricorso a numeri di mailbox differenti per ogni connessione per evitare ambiguità: infatti si è eseguito un test specifico per vedere il comportamento utilizzando sempre le mailbox 1 e 5. Il risultato faceva presupporre che la mailbox di uscita dei robot fosse condivisa o comunque che il mediatore avesse difficoltà a differenziarli poichè instradava in maniera errata il messaggio.

Un problema emerso nei test era dato dall'avere più task in parallelo nel mediatore: se si realizzavano 3 task, uno per ogni linea da controllare, non riusciva a gestirli e succedeva che ne andava in esecuzione solo uno, molto probabilmente il canale radio veniva occupato soltanto da un Thread e non dava spazio agli altri. Si è dovuto quindi utilizzare un unico task che ad ogni ciclo controllasse le 3 linee sequenzialmente. Infine, poichè le liberie erano in realtà un'applicazione si è proceduto con il dividere la parte di codice che rappresentava la libreria da quella, invece che era solo il task che doveva andare in esecuzione per ogni

specifico NXT. Questa operazione è stata eseguita senza riscontrare alterazioni o problemi del comportamento a run-time.

Tuttavia non si è riusciti ad avere solamente un header file in maniera tale da non rendere visibile l'implementazione e si è deciso di utilizzare la libreria così com'era lasciandola come file *.nxc*.

Successivamente, a libreria già ultima, è stato realizzato il sistema dinamico di assegnamento delle mailbox e di registrazione: i test seguenti hanno evidenziato la necessità di aggiungere in questa fase nel codice numerose *Wait*, tra lo scambio iniziale di messaggi con i vari robot. Infatti capitava che venisse registrato solo il primo mentre i restanti 2 non facevano in tempo a riferirlo e rimanevano esclusi dalla comunicazione.

In aggiunta si è notato, dopo numerose prove, che è meno frequente che i messaggi vadano persi se prima di effettuare una *SendRemoteString* (sfruttata anche dalle nuove API) si attende che lo stato Bluetooth del ricevente non dia errori, verificandolo con un'apposita funzione (*BluetoothStatus*).

4.6.2 Coordinazione

Una volta realizzata la libreria la si è testata aggiungendo all'applicazione di ogni singolo robot un carico più pesante (cioè inserendo più task in parallelo) e vedendo più concretamente gli effetti. Si è pensato quindi di mettere in gioco anche fattori di percezione dell'ambiente con una conseguente risposta tramite l'attivazione degli output. Inoltre si voleva provare l'invio di singoli messaggi, non ripetuti, cioè senza aver bisogno di effettuare un loop di invio o di ricezione.

Si sono posizionati i 3 robot davanti ad una parete: all'avvio, partiva il primo e non appena rilevava la presenza del muro, si fermava e mandava un messaggio al secondo, che a sua volta azionava i motori e avanzava fino a quando anch'esso incontrava lo sbarramento, e così anche per il terzo, che alla fine ricomunicava al primo che era bloccato. Quest'ultimo allora tornava indietro per qualche secondo e poi riferiva al secondo che aveva finito, e così via: il risultato è una sequenza di movimenti in cui i 3 robot si alternano.

Tuttavia si è notato che a volte i messaggi si perdono e non giungono. Per avere maggiore probabilità di successo occorre inserire dei tempi

di attesa abbastanza lunghi (più di 1 secondo) tra una Send e l'altra. Un altro fattore che può influenzare la perdita di dati è lo stato di carica delle batterie, che si ripercuote sulla reattività del robot.

4.6.3 Performance

Per quanto riguarda la libreria si sono attuate delle prove per vedere le reali prestazioni, misurate in base al calcolo del ritardo dall'invio di un messaggio all'effettiva ricezione.

La prova consiste nell'invio da parte di un robot di un messaggio a se stesso. In questo modo può, una volta effettuata una Send, segnarsi il tempo dato dal suo orologio interno di sistema per poi confrontarlo con quello al momento della ricezione e calcolare infine il delay. Sarebbe più adatto inviare un messaggio ad un robot diverso, perchè dover cambiare immediatamente modalità da trasmettitore a ricevente potrebbe causare un ulteriore ritardo, tuttavia è impossibile operare in un altro modo dal momento che i clock dei vari Brick sono indipendenti e quindi non sono sincronizzati tra loro.

Send e Receive si susseguono immediatamente, inoltre vengono invocate direttamente dal main dell'applicazione, questo per avere una situazione con meno carico possibile (un solo task attivo).

Si sono attuate 10 prove sia utilizzando la Receive che la Receive bloccante poi si sono segnati i valori ottenuti (in ms).

Sono state eseguite in situazioni diverse: prima di tutto collegando quel solo robot al mediatore, poi aggiungendoci il secondo, ed infine il terzo. I programmi che venivano messi in esecuzione sugli altri robot non svolgevano alcun tipo di attività, a parte quella di registrarsi al mediatore.

Receive	<i>1 robot</i>	<i>2 robot</i>	<i>3 robot</i>
<i>Prova 1</i>	42	96	516
<i>Prova 2</i>	36	90	486
<i>Prova 3</i>	42	66	504
<i>Prova 4</i>	54	108	546
<i>Prova 5</i>	66	96	516
<i>Prova 6</i>	54	72	522
<i>Prova 7</i>	42	64	516
<i>Prova 8</i>	36	102	486
<i>Prova 9</i>	42	84	480
<i>Prova 10</i>	48	96	492
<i>Media</i>	46,5	89,4	506,4

BlockingReceive	<i>1 robot</i>	<i>2 robot</i>	<i>3 robot</i>
<i>Prova 1</i>	73	289	1687
<i>Prova 2</i>	104	249	1757
<i>Prova 3</i>	57	235	1569
<i>Prova 4</i>	56	219	1690
<i>Prova 5</i>	27	223	1555
<i>Prova 6</i>	62	299	1883
<i>Prova 7</i>	81	221	1592
<i>Prova 8</i>	184	291	1721
<i>Prova 9</i>	92	180	1560
<i>Prova 10</i>	182	178	1566
<i>Media</i>	91,8	238,4	1658

I test mettono in evidenza che con l'aumentare di robot collegati i tempi si dilungano: questo fa pensare che il mediatore perda molto tempo nell'interrogare più device, anche se questi non stanno facendo niente di particolare. La Receive impiega mediamente 506 ms; per quel che riguarda la Receive bloccante, invece, i tempi si allungano molto: si parla in media di circa 1,6 s.

I test sono stati tutti eseguiti collegando il robot campione alla linea 1. Si è notato che, eseguendo altre prove utilizzando invece la linea 3 e confrontando i valori ottenuti, si otteneva un risultato totalmente differente.

Receive	<i>3 robot (linea 1)</i>	<i>3 robot (linea 3)</i>
<i>Prova 1</i>	516	246
<i>Prova 2</i>	486	240
<i>Prova 3</i>	504	228
<i>Prova 4</i>	546	234
<i>Prova 5</i>	516	240
<i>Prova 6</i>	522	246
<i>Prova 7</i>	516	228
<i>Prova 8</i>	486	264
<i>Prova 9</i>	480	246
<i>Prova 10</i>	492	258
<i>Media</i>	506,4	243

BlockingReceive	<i>3 robot (linea 1)</i>	<i>3 robot (linea 3)</i>
<i>Prova 1</i>	1687	725
<i>Prova 2</i>	1757	723
<i>Prova 3</i>	1569	677
<i>Prova 4</i>	1690	704
<i>Prova 5</i>	1555	816
<i>Prova 6</i>	1883	722
<i>Prova 7</i>	1592	745
<i>Prova 8</i>	1721	715
<i>Prova 9</i>	1560	748
<i>Prova 10</i>	1566	726
<i>Media</i>	1658	730,1

La comunicazione attraverso la linea 3 sembra risultare molto più veloce rispetto alla linea 1, quindi si presuppone che il mediatore abbia più difficoltà ad effettuare la ricezione sulle prime linee su cui fa il polling (si ricorda che viene eseguito sequenzialmente a partire dalla prima linea).

Quindi, anche se le librerie permettono all'utente di non preoccuparsi a livello logico delle connessioni hardware, in realtà, questi dettagli possono avere effetti sulle performance.

Ovviamente le librerie sono soprattutto utili per creare un sistema con 3 robot anche se c'è la possibilità di averne di meno (ma non sarebbe vantaggioso), e quindi interessa più che altro questo tipo di situazione,

che purtroppo risulta aver tempi di risposta molto lunghi.

Si può concludere quindi che i ritardi sono influenzati da tre fattori: la connessione hardware, il numero di dispositivi connessi ed il tipo di primitiva usata nel ricevere.

Preso atto di queste considerazioni bisogna fare attenzione nell'invio dei messaggi pensando al tempo che impiegano a giungere. Non è quindi saggio creare un'applicazione in cui un Brick possa utilizzare delle Send con un'alta frequenza, ossia occorre che il tempo che intercorre tra l'uso di una Send e l'altra sia maggiore rispetto al tempo che si impiega mediamente per ricevere un messaggio, perchè altrimenti questo ha più probabilità di andare perso.

4.6.4 Motori e sensori

Ci sono alcuni aspetti che non riguardano la comunicazione che è comunque necessario sapere per poi riuscire a realizzare un'applicazione in cui i robot agiscono cooperativamente.

Le possibilità a livello teorico che offre l'hardware di realizzare un'applicazione in cui i robot si muovono o percepiscono segnali dall'ambiente esterno sono ampie ma in realtà bisogna capire fino a che punto gli input e gli output siano affidabili e precisi, altrimenti si possono avere effetti indesiderati, che tuttavia non dipendono dalla programmazione ma esclusivamente dall'hardware che risulta spesso essere molto approssimativo nelle misure.

A questo scopo, anche per casi di studio che si vedranno più avanti, si sono eseguiti alcuni test per avere un'idea dei limiti dei sensori e dei motori sotto il profilo della precisione per poi cercare di rimediare nel migliore dei modi a livello software o con piccoli accorgimenti.

Si è riscontrato, per esempio, che nella maggior parte dei casi anche se il robot ha il compito di proseguire dritto esso in realtà devia abbastanza significativamente non riuscendo a mantenere una traiettoria rettilinea. Facendo però ricorso ad API NXC più specifiche (es. *OnFwdSync*) è possibile far sincronizzare tra loro i due motori che azionano le ruote in modo tale da attribuire ad ognuno lo stesso numero di giri e di potenza. Tuttavia anche con questo meccanismo si è riscontrato che i robot, soprattutto dopo tragitti abbastanza lunghi, alla fine comunque un po' deviano da una parte e non riescono a ri-

manere perfettamente in linea.

I motori spesso sono imprecisi anche sulla distanza: attribuendo lo stesso valore di potenza e facendo andare avanti i robot per un tempo di 6 secondi, si è visto alla fine una differenza di molti centimetri nel percorso effettuato dai vari NXT. Questo vale anche se si preferisce controllare i motori utilizzando i gradi di rotazione: con il medesimo valore, vi sono differenze a volte molto sostanziali tra i giri effettuati dai motori dei diversi robot.

Inoltre la potenza dei motori spesso è relativa: essa infatti può dipendere spesso da fattori non prevedibili come il livello di batterie ed una variazione di quest'ultimo può alterare la reale potenza erogata che risulterebbe quindi differente da quella dichiarata. Bisogna considerare poi che le pile si esauriscono molto rapidamente, e quindi risulta essere una situazione molto frequente.

Tutto questo porta a sconsigliare di realizzare applicazioni in cui i robot debbano mantenere perfettamente una stessa traiettoria o programmi in cui un NXT debba eseguire un percorso e posizionarsi esattamente in un punto senza l'ausilio di alcun sensore.

Anche la percezione dei sensori è molto variabile: per esempio utilizzando lo stesso programma su 2 differenti robot a volte poteva succedere che il sensore ad ultrasuoni rilevasse un ostacolo anche se in realtà non vi era alcun oggetto, mentre nell'altro sembrava funzionare correttamente. Una causa potrebbe attribuirsi al rumore (disturbi elettrici) o anche alla lunghezza del cavo che collega il Brick al sensore: infatti è capitato di ottenere effetti differenti sostituendolo.

Il sensore di luce è molto carente sotto il punto di vista della precisione e dell'affidabilità. Per poterlo utilizzare sono stati necessari test preliminari per riuscire a trovare il modo migliore di fargli misurare l'intensità luminosa.

Si sono eseguite alcune prove facendo leggere in modalità 'RAW' al sensore la tonalità di grigio di una striscia nera, una blu e una rossa con apparizione a video del risultato. Il nero è quello che riesce a distinguere maggiormente, mentre confonde spesso il verde con il blu, mentre per il rosso non dà particolari problemi, a parte il fatto che interpretava lo stesso risultato ogni volta che incontrava una riga grigio chiaro della suddivisione delle mattonelle del pavimento. In modalità 'RAW' il valore acquisito può variare da 0 (scuro) a 1023 (chiaro),

tuttavia i colori che si sono presi in considerazione anche se ad occhio nudo sembravano molto diversi, avevano una differenza di valori pari a volte neanche a 30, quindi bastava una minima variazione di luce per fare interpretare in maniera errata il tipo di colore. Uno tra i fattori principali è la luce ambientale: è vivamente consigliabile tenere attivo il led rosso, ma anche con quest'ultimo acceso sarebbe necessaria una luce uniforme e costante; anche per la stessa tonalità da rilevare, ma posizionata in luoghi più in ombra, la messa a punto effettuata precedentemente con un certo tipo di ambiente diviene completamente inesatta.

Si è provato a fare anche modifiche alla posizione del sensore: più in alto subisce maggiormente l'influenza della luce ambientale, tuttavia posizionandolo troppo in basso può capitare che sia talmente vicino alla superficie, da farsi ombra da solo, e quindi rilevare una tonalità scura, come se fosse nero, per tutti i tipi di colore.

Da questi test si può concludere che è preferibile creare applicazioni in cui il sensore di luce debba riconoscere pochi colori (al massimo 3) e che essi siano molto diversi tra loro e di non utilizzare ambienti con un'illuminazione non uniforme.

4.6.5 Problemi riscontrati

Oltre alle difficoltà incontrate nei vari test descritte precedentemente ve ne sono altre che comunque sono emerse durante i diversi esperimenti.

In primo luogo bisogna sottolineare la continua necessità di avere pile cariche: ogni robot, nonostante sia alimentato da 6 pile ricaricabili, potrebbe esaurirle tutte completamente nell'arco di poche ore. Le maggiori operazioni che portano ad un rapido consumo delle pile sono l'utilizzo dei motori e della tecnologia Bluetooth (specialmente lo scambio di messaggi). Non è molto consigliabile comprare la pila al litio, anche se garantirebbe un'alimentazione più duratura, perchè essa dovrebbe attaccarsi sotto al Brick: la maggior parte di tipologie di robot non avrebbe lo spazio sufficiente per farla stare. Bisogna inoltre considerare anche che per sostituire ogni volta le pile occorre aprire il Brick nella parte inferiore: questo può risultare assai difficoltoso ed a volte significherebbe smontare momentaneamente intere parti del

robot che risiedono dietro al corpo principale.

Uno tra i maggiori problemi che esula dalla programmazione ma dipende solo dalla qualità dell'hardware rimane comunque il fatto che non tutti i Brick rispondono allo stesso modo, anche con la medesima applicazione: alcuni sensori elaborano i dati in maniera più superficiale, ed anche se i dati dall'esterno dovessero essere acquisiti correttamente, alcuni Brick sembrano non reagire, cioè non attuano la procedura che dovrebbero eseguire, mentre su altri Brick viene seguita normalmente, senza vere alcun tipo di problema.

Capitolo 5

Casi di studio

In questo capitolo sono descritti due casi di studio utili per testare le potenzialità e l'efficienza delle librerie appena realizzate, si considereranno alcune situazioni particolari per vedere come i robot reagiscono e collaborano tra di loro.

5.1 Introduzione

Utilizziamo ora le API che fornisce la nuova libreria per raggiungere un particolare obiettivo in cui i robot si devono coordinare tra di loro ed eseguire determinate azioni.

Inizialmente, durante la fase di implementazione della libreria si sono sfruttate esclusivamente visualizzazioni dei messaggi sullo schermo LCD dei vari Brick NXT, senza mettere in gioco nessun altro tipo di output (motori) ed input (sensori), a parte piccoli casi.

Si vogliono ora mettere in mostra comportamenti cooperativi basandosi su movimenti, azioni e percezioni nel territorio circostante. Si penseranno a casi di studio per vedere effettivamente come i robot reagiscono e riescono poi a coordinarsi grazie alla nuova libreria.

5.2 Interazione fra robot: coordinazione nei movimenti

Nel primo caso di studio un NXT ha il ruolo di caposquadra ed impartisce ordini ai restanti 2 (il quarto funge solo da mediatore della comunicazione).

Il caposquadra si sposterà nello spazio circostante secondo una serie di movimenti prestabiliti: nel frattempo comunicherà agli altri il comando con lo scopo di far eseguire anche all’NXT2 e all’NXT3 gli stessi movimenti con il risultato che tutti e tre i robot sembreranno muoversi contemporaneamente allo stesso modo.

Tuttavia si aggiunge anche la possibilità di un imprevisto, che possa fare in modo di non avere solo una sequenza di comandi prestabilita: il caposquadra ha attivo il sensore di distanza in modo tale che se dovesse incontrare un ostacolo, abbia poi la possibilità di comunicare agli altri di fermarsi o comunque di eseguire una procedura di movimenti diversa, per poterlo evitare.

Nella prova eseguita i 3 robot erano disposti in fila, con in testa il caposquadra, l’unico a conoscere la sequenza dei movimenti.

I robot utilizzati sono stati l’Alpha Rex¹ per il mediatore (NXT0) e TriBot² per la squadra di robot che si deve muovere nello spazio (NXT1, NXT2, NXT3). Questa scelta è stata adottata per avere fisicamente una distinzione, in modo tale da riconoscere immediatamente le 2 tipologie di ruoli. Per i primi esperimenti non era stato assolutamente significativo il tipo di robot perchè si sfruttava solo lo schermo, tuttavia per il loro grado di mobilità è opportuno ora utilizzare i TriBot. In questi ultimi, però, sono state apportate alcune modifiche: sono state eliminate le tenaglie per avere più libertà nei movimenti ed è stato spostato il sensore di luminosità sul dorso, per poterne utilizzare la luce come spia.

¹Appendice C.4 - Tipologie di robot NXT

²Appendice C.1 - Tipologie di robot NXT

5.2.1 Risultati ottenuti

Una volta attivati i programmi, e dopo essersi tutti e 3 registrati al mediatore come di consuetudine, l'NXT1 (il caposquadra) inizia a muoversi: prima comunica cosa ha intenzione di fare agli altri 2 robot, poi esegue lui stesso. Questo è stato inevitabile per poter far in modo che tutti e 3 si spostassero contemporaneamente: i tempi di invio e di risposta dei messaggi scambiati sono abbastanza lunghi (come dimostrato dai precedenti test) e occorre qualche istante affinché essi vengano processati.

I messaggi contengono un'informazione che rappresenta una tipologia di comando (es. 'avanti', 'destra', 'gira', ecc.). Per ogni comando gli NXT sono stati 'calibrati' individualmente con particolari livelli di potenza ed angolatura della rotazione dei motori per permettere che eseguissero più o meno lo stesso movimento, perchè come è risultato nelle prove precedenti, anche con valori uguali i Brick reagiscono comunque in maniera leggermente diversa.

Come prima azione l'NXT1 va avanti, ma incontra quasi subito un ostacolo posizionato poco più avanti: un blocco che impedisce il passaggio.



Figura 5.1: Rilevamento di un ostacolo

Una volta percepita la sua presenza grazie al sensore, ferma subito gli altri, inviando loro un messaggio, poi si blocca lui stesso ed infine

esegue una procedura che solo esso conosce (quando entra in questa modalità attiva il led rosso del sensore di luminosità), con la quale grazie ad una serie di movimenti tenta di aggirarlo; gli altri 2 robot eseguiranno le stesse azioni.



Figura 5.2: *Aggiramento dell'ostacolo*

Tuttavia anche se uno degli altri 2 robot in coda dovesse incontrare improvvisamente un ostacolo, attiverebbe la luce rossa e si bloccherebbe, successivamente manderebbe un messaggio al caposquadra che a quel punto si blocca lui stesso e ferma anche il restante NXT.

5.2.2 Considerazioni

Avendo comunque un robot che fa da punto di riferimento e perno centrale, lo si può paragonare al sistema nativo formato da Master e Slave, tuttavia questa prova è stata realizzata esclusivamente per testare la reazione dei robot alle nuove API.

Inoltre esistono già primitive NXC che permettono di controllare direttamente un altro robot, impartendogli comandi; la differenza risiede nel fatto che i messaggi comunque non sono semplici ordini, ma informazioni, che ogni robot interpreterà o saprà sfruttare come ritiene opportuno (dipende ovviamente da come è stato implementato il programma).

Questa risulta quindi una semplice prova di coordinazione, basata sull'utilizzo della nuova libreria, ma non mette in evidenza tutti i vantaggi che derivano dal loro impiego.

I robot sono riusciti a comunicare in maniera diretta ed efficace, secondo il pattern del Message Passing diretto, e questo avviene senza preoccuparsi dell'hardware.

Tuttavia il protocollo sviluppato pecca ancora di affidabilità: è facile che messaggi vadano persi perchè non vi sono meccanismi che assicurino che essi siano stati ricevuti.

Ciò può succedere spesso sia perchè il tragitto comunque è lungo (possono incontrare errori sia nella trasmissione al mediatore, sia nella ritrasmissione al destinatario), e sia perchè l'hardware di base dei robot NXT sotto il profilo della comunicazione Bluetooth lascia spesso a desiderare, tuttavia questa libreria non ha il compito di risolvere questi problemi.

Si è anche notato che avendo più task in esecuzione su un unico robot, è più probabile che i messaggi non giungano: lo scheduler infatti ha più difficoltà a processarli, se è impegnato a gestire più attività contemporaneamente.

5.3 Interazione fra robot: esplorazione

A differenza del primo caso di studio, il secondo mette in gioco forme di comunicazione e coordinazione peer-to-peer: i robot devono esplorare il territorio in cerca di determinati target.

L'obiettivo di questo caso di studio è mettere in luce che non esiste più una gerarchia di ruoli: tutti i robot hanno il medesimo compito e possono comunicare in qualsiasi istante (il ritrovamento dei punti di riferimento è casuale) con chiunque.

L'area da esplorare sarà delimitata da una linea nera, mentre i punti da trovare saranno rappresentati da alcune macchie colorate sul pavimento; per questo genere di prova verranno utilizzati sempre robot di tipo TriBot senza tenaglie e con il sensore di luce posizionato in modo tale da far percepire meglio la luminosità dei colori del suolo.

I robot dovranno andare in giro evitandosi grazie al sensore ad ultrasuoni e dovranno spostarsi per trovare ognuno un punto di riferi-

mento. Dovranno comunque rimanere dentro l'area, ed abbandonarla solamente quando tutti avranno eseguito il loro compito.

5.3.1 Comportamento dei robot

I robot, posizionati casualmente all'interno del territorio, dopo la registrazione al mediatore, iniziano a muoversi abbastanza lentamente. Al rilevamento della linea nera, rappresentante il bordo dell'area da perlustrare, eseguono una manovra in cui tornano leggermente indietro e poi ruotano di un certo numero di gradi casuale. Se, grazie al sensore ad ultrasuoni, rilevano un ostacolo (in questo caso rappresentato da un altro robot NXT), eseguono lo stesso procedimento per evitarlo. Un robot riconosce di aver raggiunto l'obiettivo se rileva con il sensore di luce una tonalità molto più scura del pavimento, ma più chiara del nero e a quel punto si ferma, spegnendo il led rosso, poi manda un messaggio agli altri 2 robot riferendo che ha trovato qualcosa, ed infine rimane in attesa.

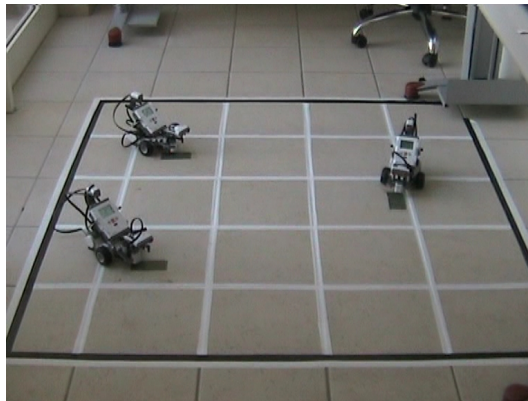
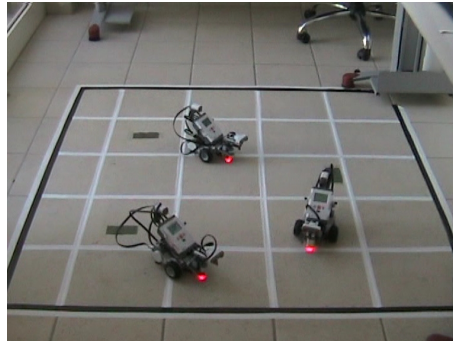
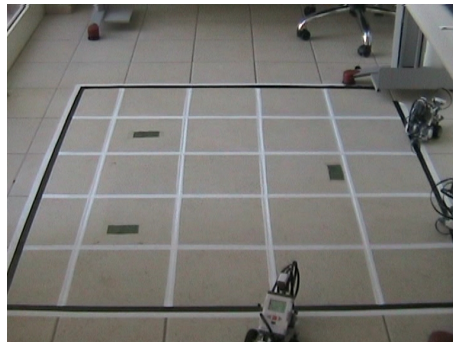


Figura 5.3: *Raggiungimento dell'obiettivo*

Ogni robot tiene memoria del numero degli obiettivi trovati complessivamente e quando anche l'ultimo robot arriva a destinazione, portando a termine il proprio compito, riattivano il led rosso e si mettono in movimento per abbandonare l'area: una volta superata la linea nera si bloccano automaticamente.

Figura 5.4: *Riattivazione*Figura 5.5: *Abbandono dell'area*

In generale l'esperimento ha avuto successo, tuttavia per arrivare ad un risultato soddisfacente è stato necessario fare numerosi tentativi di calibrazione del sensore di luce e pianificare bene i tipi di movimenti da eseguire in caso di rilevamento di un ostacolo o della linea nera: i maggiori problemi derivavano principalmente da queste cause. La comunicazione nella maggior parte dei casi aveva esito positivo, solo poche volte è capitato che alcuni messaggi non giungessero al destinatario.

5.3.2 Considerazioni

Questo caso di studio, al contrario del primo, mette in risalto le potenzialità della libreria: il ritrovamento di un colore è qualcosa di casuale,

poichè non si saprà chi riuscirà a trovarlo per primo; si mostra che non c'è niente di prestabilito, e chiunque, dopo aver eseguito il proprio compito, può inviare messaggi a tutti, in modo tale eliminare la distinzione tra Master e Slave, poichè sono tutti sullo stesso livello.

Inoltre non viene ordinato di eseguire immediatamente qualcosa, ma viene comunicata solo un'informazione che permetterà ad ogni robot di conoscere lo stato della situazione e rimanere quindi aggiornato.

Capitolo 6

Conclusioni

Si è avuto l'opportunità di studiare il comportamento di questi particolari robot e di agevolarne l'uso sotto il profilo della comunicazione: questo permetterà di avere una base software per non doversi preoccupare di problemi di più basso livello che daranno poi la possibilità di realizzare in futuro esperimenti più complessi che possono interessare vari ambiti.

È interessante aver unito lo studio e l'approfondimento di modelli ricorrenti anche in architetture e sistemi più complessi (Internet, Web Services, ecc.) come il Message Passing, con questi tipi specifici di device, usati come prototipi per qualsiasi genere di esperimento.

Il comportamento di tali robot non è lontano da quello che poi si può applicare alla robotica di più alto livello (per esempio utilizzata in ambito industriale o scientifico), poichè si è tentato di studiare e proporre un rimedio esclusivamente dal punto di vista software, non importa poi come siano stati realizzati fisicamente i robot.

Con la realizzazione di questa libreria è stato raggiunto un maggior grado di astrazione: ha permesso di non preoccuparsi dei vincoli dati dall'hardware di base, inoltre un approccio basato sulla comunicazione diretta agevola all'utente numerose operazioni.

Tuttavia si potrebbero apportare ancora diversi miglioramenti anche se il passo più importante è stato fatto ed è già molto soddisfacente.

6.1 Sviluppi futuri

Avere una libreria che permetta di far comunicare direttamente due NXT qualsiasi senza preoccuparsi dei collegamenti hardware è già un notevole progresso nel protocollo di comunicazione di questi robot, ma lo si può considerare solo un punto di partenza: si possono ancora compiere notevoli passi in avanti.

6.1.1 Miglioramento dell'affidabilità

Gli sviluppi futuri a partire dalla libreria già realizzata sono legati a problemi di affidabilità.

Per garantire che un messaggio giunga a destinazione bisognerebbe l'introdurre gli ACK (Acknowledge): questo potrebbe essere un successivo miglioramento da sviluppare in futuro, tuttavia si hanno dei dubbi sui reali vantaggi dopo una realizzazione di un sistema basato su questo meccanismo e se ne valga veramente la pena. Infatti si nota già che anche con il solo metodo utilizzato finora, i tempi di risposta sono abbastanza lunghi, se si introducessero nuovi messaggi, che inoltre non contengono nessuna informazione utile, si ridurrebbe ulteriormente l'efficienza. Inoltre intasando la rete di messaggi può darsi che non solo si allunghi maggiormente l'attesa, ma potrebbe verificarsi un numero elevato di errori, anche se poi il protocollo permetterebbe di rimediare (ma questo significherebbe ancora più messaggi da ritrasmettere).

Bisogna considerare anche che sarebbe difficile realizzare un meccanismo in cui si abbia la garanzia di aver ricevuto anche gli ACK, poichè anch'essi sono alla fine dei messaggi, seppur privi di informazioni utili; per esempio si consideri il caso che un robot invii un messaggio: il destinatario lo riceve e poi rimanda un ACK di conferma, ma quest'ultimo si perde e non giunge a destinazione. Il mittente non ha opportunità di poter sapere se lo ha ricevuto, l'unica cosa che può fare, non avendo ricevuto niente, è rimandare di nuovo il messaggio e questo significherebbe comunicare al destinatario 2 volte la stessa cosa. Occorrerebbe quindi adottare politiche appropriate per evitare tutto questo.

Come altro sviluppo si potrebbe anche realizzare la Send bloccante: esiste in NXC una variabile che permette di sapere se una determinata

mailbox è vuota, pertanto non è risultato complicato realizzare la Receive bloccante, che attende fin quando non vi è un nuovo messaggio. Risulta tuttavia assai più complicato nel caso della Send: non vi sono modi di sapere se la mailbox è piena, cioè se ha già raggiunto il limite massimo di 5 messaggi. In questo caso si dovrebbe bloccare fin quando non vi è spazio disponibile, tuttavia succede in realtà che sovrascrive il più vecchio e continua. Il metodo forse più efficace sarebbe fare in modo che la mailbox possa contenere un solo messaggio e quindi avere la possibilità di inserirne uno nuovo esclusivamente se in quel momento è vuota. Ovviamente poter avere un singolo messaggio per mailbox nella maggior parte dei casi d'uso non è una soluzione molto performante, perchè limita notevolmente la possibilità di comunicazione.

6.1.2 Comunicazione indiretta

Un interessante progresso sarebbe riuscire ad avere come mediatore il pc. In questo modo si avrebbero numerosi vantaggi, tra cui maggiore capacità di memoria disponibile e maggiore velocità di elaborazione.

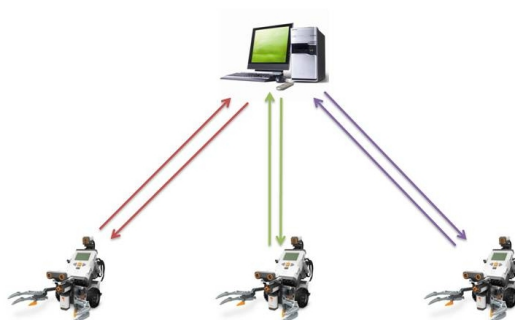


Figura 6.1: *Interfacciamento con il pc*

Partendo da questa base non sarebbe poi complicato realizzare un modello di comunicazione indiretta basato sugli spazi. Il pc sarebbe infatti l'ideale per accogliere e far risiedere un numero considerevole di informazioni (le tuple) per la quantità di memoria disponibile, poichè ciò sarebbe impossibile da realizzare con le sole mailbox messe a disposizione dai robot NXT, ma soprattutto, il reale vantaggio risiederebbe

nell'avere una maggior rapidità di calcolo che permetterebbe di supportare la gestione di questo complicato sistema.

Per poter attuare ciò occorre orientarsi verso un linguaggio java-like: come descritto nel primo capitolo, la soluzione migliore sarebbe utilizzare la libreria iCommand per l'applicazione che risiede su pc, anche perchè non vi è modo di crearne una basata sul linguaggio NXC, che tuttavia potrà comunque essere sfruttato per implementare i programmi che verranno eseguiti sui vari Brick NXT.

Apparentemente si avrebbe uno schema simile a quello di Master e Slave, tuttavia a livello logico è molto differente: il pc rimane un'unità passiva (è lo spazio) in cui i robot possono immettere e prelevare informazioni, astraendo dal tempo, dalle connessioni, dai destinatari, dai mittenti, ecc.

Avere il computer come quarto device ovviamente è utile solo nel caso esso abbia la funzione di Master: collegato come Slave non porterebbe a nessun beneficio significativo.

Appendice A

RCX

A.1 La prima generazione di mattoncino intelligente

L'*RCX* (Robot Command eXplorer), distribuito commercialmente con il nome *RIS* (Robot Invention System), è la prima serie prodotta della linea LEGO Mindstorms. L'*RCX* originale è stato rilasciato nel 1998 ed è stato sostituito poi nel 2006 dagli *NXT*.



Figura A.1: *RCX* e sensori

Esiste anche un altro modello più semplice di *RCX* denominato *Scout*, con 2 porte di input, 2 di output, un sensore di luce integrato e senza interfaccia con il pc; inoltre poteva memorizzare un solo programma.

Le principali caratteristiche tecniche dell'RCX sono:

- Microprocessore a 8/16 bit Hitachi H8/3292, chiamato H8/300 CPU Core a 16 MHz
- 16k di Memoria ROM (interna, sul chip)
- 512b di Memoria RAM interna (on-chip)
- 32k di Memoria RAM esterna (on-board)

Connettività:

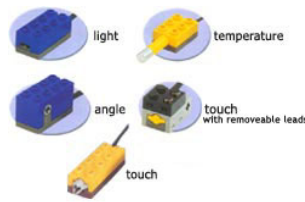
- Porta ad Infrarossi (IR)
- Ingresso per alimentazione esterna continua (solo nella versione 1.0)
- 3 porte di input
- 3 porte di output (con convertitore A/D da 10 bit)

Altre caratteristiche:

- 4 pulsanti.
- Alimentazione a batterie
- Display LCD bianco e nero composto da 5 digits
- Speaker

A.2 Tipi di sensori

- *Tatto*: sensibile alla pressione
- *Luce*: misura l'intensità luminosa percependo anche gli Infrarossi
- *Rotazione*: misura rotazioni o angoli percependo gli incrementi di 22,5 gradi
- *Termico*: misura la temperatura e può lavorare sia con i gradi Celsius che con i Fahrenheit in un intervallo che va dai -20° C ai +70° C (dai -3° F ai +157° F)

Figura A.2: *Sensori*

A.3 Altri componenti forniti

L'unico modo per permettere lo scambio dati con altri dispositivi è tramite la porta ad Infrarossi. Con il pc il traferimento avviene grazie all'ausilio di una torretta, l'*IR Tower*, che trasmette (o riceve) all'RCX ad una distanza massima di 10 metri. Il protocollo IR utilizzato dal Brick LEGO non è tuttavia standard e potrebbe non essere compatibile con tradizionali dispositivi ad Infrarossi.

Un altro accessorio compreso nel kit è un telecomando per il controllo a distanza.

A.4 Linguaggi di programmazione

I linguaggi di base forniti dalla LEGO per questo minicomputer sono grafici: RCX Code e ROBOLAB (basato su LabView).

Altri linguaggi sono: C e C++ (con sistema operativo BrickOS, precedentemente LegOS o utilizzando QuiteC), Java (con leJOS o TinyVM), NQC, Lego.NET, OnScreen, pbFORTH, PbrickDev, PRO-BOT, SqLEGO, TclRCX, TerrapinLOGO, Visual Basic, XS, actor-lab (grafico).

Appendice B

Linguaggi di programmazione per NXT

I Brick NXT possono essere programmati tramite differenti tipi di linguaggi, tra i generi più diffusi. Se lavorano con il firmware standard hanno la possibilità di funzionare direttamente sul Brick, altrimenti bisognerà installare un nuovo firmware che funga da Virtual Machine, affinché il codice venga interpretato. La compilazione tuttavia è data a programmi sviluppati da terzi, poichè la LEGO fornisce un supporto alla programmazione e compilazione esclusivamente grafico (NXT-G) e non tramite la scrittura di codice.

B.1 Caratteristiche

Di seguito vengono illustrate le principali caratteristiche e potenzialità di alcuni tra i più diffusi linguaggi di programmazione dei Brick NXT. Inoltre verrà messa anche una tabella che consiglia il tipo di linguaggio a seconda dell'uso ed degli scopi da raggiungere.

APPENDICE B. LINGUAGGI DI PROGRAMMAZIONE PER NXT

Features	NXT-G Retail	NXT-G Educational	RoboLab 2.9	NBC	NXC	RobotC	NI LabVIEW Toolkit	leJOS NXJ	pbLua	LEJOS OSEK
Language type	Graphic	Graphic	Graphic	Assembly	C-like	C	Graphic	Java	Lua	ANSI C
Firmware	Standard	Standard	Standard (#1)	Standard	Standard	Standard (#1)	Standard	Custom	Custom	Custom
IDE (included?)	Yes	Yes	Yes	Yes	Yes	Yes	No (#6)	plugins for Eclipse and NetBeans	No (#7)	Eclipse CDT (GCC+ATMEL SAM-BA)
Windows	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes (#7)	Yes
Mac OSX	Yes	Yes	Yes	Yes	Yes	Not Yet	Yes	Yes	Yes (#7)	No
Linux	No	No	No	Yes	Yes	No	No	Yes	Yes (#7)	No (maybe)
Events	No	No	Yes	No	No	Yes	No	Standard Java events		Yes (OSEK RTOS)
Multithreading	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		Yes (OSEK RTOS)
Bluetooth Brick to PC	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Bluetooth Brick to Brick	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Not yet
Bluetooth Brick to Other Device	No	No	No	No	No	Yes	No	Yes	Yes	No
IDE Support	(#5)	(#5)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes (US only)
File System	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Not Yet	Not planned
Floating Point	No	No	Yes	No	No	Yes	No?	Yes	(#8)	Yes
Datalog	No	No	Yes	No	No	Yes	No	Yes	No	Not Yet
How to get it	Included with retail version of NXT	Available with Educational Version of NXT	Available with Educational Version of NXT	BricksCC Web Site	BricksCC Web Site	RobotC.net or LEGO Education	LabVIEW toolkit Site (#6)	Free download from leJOS Web Site	pbLua Site	LEJOS OSEK site
Support for 3rd party devices	Yes	Yes	?	?	Yes	Yes	?	Yes	?	?

Figura B.1: Confronto fra linguaggi

Features	NXT-G Retail	NXT-G Educational	RoboLab 2.9	NBC	NXC	RobotC	NI LabVIEW Toolkit	leJOS NXJ	pbLua	LEJOS OSEK
What do you want to do... (#2)										
Make robots move without learning to program	Yes	Yes	Yes							
Learn to Program using the NXT				No						
Write "Fast" programs				Yes	Yes	Yes		Maybe?	Yes	Yes
Write programs "Fast"	Yes	Yes	Yes							
Learn Advanced Programming concepts				?	Yes	Yes	Yes	Yes	Yes	Yes
Suited For (#3)	People starting with the NXT or starting with doing simple tasks	Schools upgrading from the NXT to the NXT	Schools upgrading from the RCX to the NXT	Advanced programmer	C programmer	Applications requiring maximum speed	LabVIEW users, or people wanting to improve NXT-G	Java Programmers	Educational Users	Embedded software programmer, C/C++ programmer

Figura B.2: Casi d'uso

B.2 Performance

È stato eseguito il testing su un programma scritto in linguaggi diversi ma che in ogni caso alla fine esegue le medesime operazioni. Gli autori di questo test hanno preso poi in considerazione solo alcuni aspetti per valutarne l'efficienza.

Lo pseudo-codice del programma da realizzare era:

```

Start loop
Read light sensor(3)
Read US sensor(4)
Read & Display Rotation Sensor(B)
Get random number (1-100) (RN to be used later)
Display value of:(Light Sensor+US Sensor+Rotation Sensor)*100/(RN)
Set motor speed for B and C to RN (Using Coast)
If RN > 50, Increase A by 1
If RN < 50, Decrease A by 1
If RN = 50, no change to A
Display A
Set motor A speed to A (if A<0, set direction to reverse)
Display Loop Count
Loop for 60 seconds
Stop Motors
Show display for 10 seconds
    
```

	NXT-G Retail	NXT-G Educational	RoboLab 2.9	NBC	NXC	RobotC	NI LabVIEW Toolkit	leJOS NXJ	pbLua	LEJOS OSEK
Test Program (#4)										
Tested Version	1.1	2.9							13c	
Speed (loops/min)	762	73k			4285	103k	750-5350 (#9)		18k	2695
Speed (#10)						134k-412k			31k	334k/18144k
Memory (bytes)	10704 bytes	559 bytes			1428 Bytes	561 bytes	8084/1890		1750 bytes	18240 (includes firmware)
Time to write	10 minutes	~20 min			30 minutes	30 min	15 min/30 min			30 minutes
Program	Code, Graphic	Code, Graphic			Code	Code1, Code2, Author's comments	Code, Graphic, Code1 + Code2, Graphic		Code 1 Code 2	Code
Other Comments (Facts & Opinions)	This software is designed for kids, but is not too limiting for adults	Same as Retail version, with different "Academy" robots.	LEGO has said this is the last version of RoboLab that will be made.				Can be used to create blocks that work in NXT-G programs, OR it can create programs to download directly to the NXT OR it can create PC programs to control the NXT(see below)		Early in development	LEJOS OSEK is NOT part of the leJOS project, but does use parts of leJOS code.

Figura B.3: Performance

B.3 Controllo remoto

Infine sono stati realizzati numerosi software e librerie in linguaggi diversi per permettere di comandare il Brick (soprattutto i suoi output) direttamente, in alcuni casi senza necessità di sviluppare un programma apposito.

Software	Language type	Control type	NXT Required Firmware	Link Type	Link Source	Windows	Mac OSX	Linux	Read Sensors	Websites
LEGO NXT Mobile Application	Simple RC	Remote Control	Standard (#2)	Bluetooth	Phone or PDA	-	-	-	No	LEGO
FunkNXT	Simple RC	Remote Control	Standard	Bluetooth	Phone	-	-	-	Yes	FunkNXT
BT RC	NXT-G	NXT to NXT Remote	Program running on NXT	Bluetooth	Another NXT	-	-	-	User Programmable	BTRC
Simple BT Remote	Simple RC	Remote Control	Standard	Bluetooth	Desktop	Yes	No	No	Yes	Simple Windows RC
RobotC	Simple RC	Remote Control	Standard (#1)	USB/BT	Desktop	Yes	Not Yet	No	Yes	Robot C Web Site
BricxCC	Simple RC	Remote Control	Standard	USB/BT	Desktop	Yes		No	Yes	BricxCC Web Site
OnBrick PDA	Graphic	Programmable RC	Standard	Bluetooth	PDA	-	-	-	Yes	OnBrick
OnBrick PC	Graphic	Programmable RC	Standard	Bluetooth	Desktop	Yes	No	No	Yes	OnBrick
NXT Director	Simple RC	Customizable Remote Control	Standard	Bluetooth	Palm PDA	-	-	-	No?	Director
RoboDNA	Simple RC	Remote Control	Standard	Bluetooth	Desktop	Yes			Yes	RoboDNA
MS Robotics Studio	NET	User Program running on PC	Standard	Bluetooth	Desktop	Yes			Yes	(non-commercial version is free) Download site or Microsoft Site
NI LabVIEW Toolkit	Graphic (LabVIEW G)	User Program running on PC	Standard	USB/BT	Desktop	Yes	Yes		Yes	LabVIEW toolkit Site
RoboLab	Graphic	User Program running on PC	Standard	USB	Desktop	Yes	Yes		Yes	Robolab
iCommand	Java	User Program running on PC	Standard	Bluetooth	Desktop or PDA	Yes		Yes	Yes	iCommand
LEGO::NXT	Perl	User Program running on PC	Standard	USB/BT	Desktop	Yes	Yes	Yes	Yes	Perl
nxt-Ruby	Ruby	User Program running on PC	Standard	Bluetooth	Desktop	Yes	Yes	Yes	Yes	Ruby
NXT#	C#	User Program running on PC	Standard	Bluetooth	Desktop	Yes			Yes?	NXT#
Mindsqualls	C#	User Program running on PC	Standard	Bluetooth	Desktop	Yes			Yes	Mindsqualls
NXT Python	Python	User Program running on PC	Standard	USB/BT	Desktop	Yes			Yes?	Python
My Robot Me	Graphic?	User Program running on PC	Standard	USB/BT	Desktop	Yes	No	No	Yes	This web site seems to be gone... Robot Me

Figura B.4: Software per il controllo remoto

Appendice C

Tipologie di robot NXT

I pezzi inseriti nel kit LEGO Mindstorms NXT sono predisposti per costruire uno tra le differenti tipologie di robot. Ovviamente con un po' di inventiva i pezzi sono abbastanza sufficienti per costruirne altri tipi.

Vengono messe a disposizione le istruzioni per realizzare 4 diversi robot ma sul sito ufficiale della LEGO spesso comunque vengono inserite nuove guide per realizzare altri modelli con il kit di pezzi base già fornito.

C.1 TriBot



Figura C.1: *TriBot NXT*

I *Tribot* sono in grado di spostarsi nell'ambiente tramite 2 ruote comande ognuna da un motore, il servo motore restante permette di

aprire e chiudere le chele poste nella parte anteriore, utilizzate per afferrare oggetti. Davanti, tra i bracci delle due tenaglie, sono posti il sensore di luce puntato sul terreno, quello di tatto a media altezza, ed infine in alto quello ad ultrasuoni. È stato inserito anche il sensore di rumore nella parte posteriore, sopra al Brick, che invece funge da corpo centrale.

Questo robot è il più semplice e veloce da realizzare e per questo è spesso quello che si preferisce utilizzare, inoltre il Brick è fissato a pochissimi altri pezzi del robot, quindi risulta abbastanza immediato dover staccarlo per sostituire le pile. La possibilità di avere a disposizione tutti e 4 i sensori lo rende abbastanza versatile e adattabile a qualsiasi tipo di esperimento e programma. Avendo le ruote, è applicato soprattutto quando c'è bisogno di avere un robot in grado di muoversi e spostarsi velocemente in giro per l'ambiente circostante, in esplorazione. Il sensore di luce puntato verso il basso permette di realizzare applicazioni in cui possa seguire una linea scura sul pavimento, come se fosse una guida; con il sensore di tatto riesce a rilevare la presenza di un oggetto (per esempio una palla) ed a chiudere le tenaglie, mentre quello di movimento è in grado di fargli evitare gli ostacoli o le pareti dell'ambiente in cui si trova.

C.2 RoboArm T-56

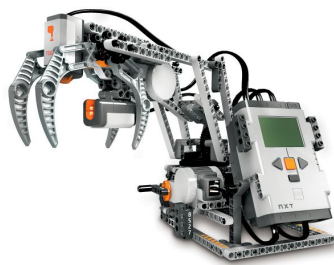


Figura C.2: *RoboArm T-56 NXT*

Il *RoboArm T-56* è un braccio meccanico; non può spostarsi dalla sua posizione perchè è fisso ma è in grado di afferrare oggetti posti sotto al braccio. Il Brick è inserito nella parte posteriore; un servo motore viene utilizzato per far ruotare il braccio e quindi farlo cambiare di posizione, mentre un altro per alzarlo (ritraendolo) ed abbassarlo (estendendolo); il terzo per stringere o lasciare la presa. In mezzo alle tenaglie alla fine del braccio è posto il sensore di tatto, mentre lateralmente quello di luce, per distinguere il colore (la luminosità) dell'oggetto afferrato. In questo robot rimangono esclusi il sensore di distanza e quello sonoro.

In dotazione sono date 2 palle, una rosa e una blu, adatte ad essere utilizzate come oggetti target del braccio; un classico programma da fargli eseguire è riconoscere la palla giusta con il sensore di luce, prenderla chiudendo le tenaglie al rilevamento con il sensore di tatto, e poi rilasciarla da un'altra parte. È adatto per tutte quelle applicazioni dove non occorre che il robot si muova e vada in giro.

C.3 Spike



Figura C.3: *Spike NXT*

Spike ha le sembianze di uno scorpione: un servo motore è utilizzato per muovere tutta la parte destra dell'animale (tenaglie e 3 zampe), ed un altro per muovere quella sinistra. L'ultimo servo motore è utilizzato per azionare il pungiglione: quest'ultimo si estende in avanti fino a toccare il suolo con il sensore di tatto posto alla sua estremità,

scavalcando tutto il dorso, formato dal Brick. Il sensore di distanza è posto davanti ed è utilizzato come testa, inoltre ha la capacità di percepire i rumori grazie al sensore posto sulla schiena, in una zona laterale. Il sensore di luce non è utilizzato in questo genere di robot. Generalmente gli si può far rilevare la presenza di qualcosa davanti a sé grazie al sensore ad ultrasuoni, in modo tale da ‘arpionarlo’ con la coda, per poi ritirarla una volta raggiunto tale scopo (quando il sensore di tatto rileva qualcosa). Non ha molte altre applicazioni, anche la sua mobilità non è eccellente e risulta abbastanza lenta.

C.4 Alpha Rex



Figura C.4: *Alpha Rex NXT*

L’*Alpha Rex* è un robot umanoide: 2 servo motori che controllano le gambe, permettono di farlo avanzare, mentre un terzo ha la capacità di far muovere contemporaneamente le 2 braccia e la testa. Il Brick forma il busto del robot; sul dorso è posto il sensore di luce, puntato verso l’alto; al posto delle mani ha in un braccio il sensore di rumore e nell’altro quello di tatto; il sensore ad ultrasuoni gli fa da testa. Avendo a disposizione tutti i sensori può essere usato per molteplici scopi. In genere il sensore ad ultrasuoni può essere efficace per rilevare la presenza di qualcosa che improvvisamente compare, quindi utilizzarlo come sensore di movimento; poichè l’Alpha Rex è abbastanza lento nel camminare, è consigliato utilizzare il TriBot nel caso si voglia avere un robot che vada in giro in esplorazione. Il sensore di luce puntato verso l’alto è in grado di rilevare se la stanza in cui

si trova è buia o comunque se vi è un cambiamento improvviso di luminosità dell'ambiente.

Elenco delle figure

1.1	<i>NXT Brick</i>	2
1.2	<i>Interno NXT Brick:</i> [1]Bluetooth, [2]Processore, [3]Coproprocessore	3
1.3	<i>Interazioni fra le varie parti del Brick</i>	4
1.4	<i>Sensore di Tatto</i>	4
1.5	<i>Sensore di Suono</i>	4
1.6	<i>Sensore di Luce</i>	5
1.7	<i>Sensore ad Ultrasuoni</i>	5
1.8	<i>Servo motore</i>	5
1.9	<i>Sensore di Colore</i>	6
1.10	<i>Preda e predatore</i>	12
1.11	<i>Avvistamento</i>	12
1.12	<i>Simulatore NXT di un diffraziometro</i>	13
1.13	<i>RFID Reader applicato ad un NXT</i>	14
1.14	<i>Primo prototipo realizzato</i>	15
1.15	<i>Rete LAN Wireless</i>	16
1.16	<i>Risolutore del cubo di Rubik</i>	18
1.17	<i>Riconoscimento dei colori</i>	18
2.1	<i>Comunicazione tra 4 NXT via Bluetooth</i>	22
2.2	<i>Diagramma di comunicazione</i>	24
2.3	<i>Interfaccia Hardware tra ARM7 (processore) e il chip BlueCore</i> .	26
3.1	<i>Message Passing diretto</i>	33
3.2	<i>Medium di comunicazione/coordinazione</i>	34
3.3	<i>Message Passing indiretto</i>	34
3.4	<i>Message Passing indiretto con mailbox condivisa</i>	35
3.5	<i>Spazio di tuple</i>	36

4.1	<i>Comunicazione tra 2 Slave</i>	40
4.2	<i>Comunicazione bidirezionale tra 3 NXT</i>	44
4.3	<i>Inizializzazione per registrare i robot al mediatore</i>	45
4.4	<i>Librerie di comunicazione</i>	46
4.5	<i>Bricx Command Center</i>	50
5.1	<i>Rilevamento di un ostacolo</i>	65
5.2	<i>Aggiramento dell'ostacolo</i>	66
5.3	<i>Raggiungimento dell'obiettivo</i>	68
5.4	<i>Riattivazione</i>	69
5.5	<i>Abbandono dell'area</i>	69
6.1	<i>Interfacciamento con il pc</i>	73
A.1	<i>RCX e sensori</i>	75
A.2	<i>Sensori</i>	77
B.1	<i>Confronto fra linguaggi</i>	80
B.2	<i>Casi d'uso</i>	80
B.3	<i>Performance</i>	81
B.4	<i>Software per il controllo remoto</i>	82
C.1	<i>TriBot NXT</i>	83
C.2	<i>RoboArm T-56 NXT</i>	84
C.3	<i>Spike NXT</i>	85
C.4	<i>Alpha Rex NXT</i>	86

Bibliografia

- [1] The annual UOIT engineering robotics competition. <http://www.engineering.uoit.ca/roboticscomp/>.
- [2] Breedbot: il robot da allevare. Technical report, Intelligent Systems Laboratory, Università degli Studi di Napoli Federico II. <http://www.isl.unina.it/>.
- [3] Bricx Command Center 3.3. <http://bricxcc.sourceforge.net>.
- [4] Embodied AI and High-Level cognitive skills. Technical report, University of Otago. <http://www.cs.otago.ac.nz/cosc451/lego.html>.
- [5] LEGO Mindstorms official web site. <http://mindstorms.lego.com/>.
- [6] LeJOS - java for LEGO Mindstorms, iCommand. <http://lejos.sourceforge.net/>.
- [7] NXTASY.ORG community. <http://www.nxtasy.org/>.
- [8] RXTX. <http://www.rxtx.org/>.
- [9] THE NEXT STEP - LEGO Mindstorms NXT blog. <http://thenxtstep.blogspot.com/>.
- [10] Vernier software & technology global gateway. <http://engineering.vernier.com/k-12/>.
- [11] Breedbot: robot ‘geneticamente’ selezionati. *Newton*, Dicembre 2005.
- [12] A. Almatrouk, A. Bermudez, and J. Kakinami. *RFID reader/sensor for LEGO Mindstorms NXT*. PhD thesis, University of San Diego, 2007 May.
- [13] C. Becchi, S. Belotti, E. Colò, and P. Silvia. *Studio sulla comunicazione infrarossa fra robot LEGO Mindstorms mediante ambiente operativo LeJOS*. PhD thesis, Università degli Studi di Genova, Ottobre 2004.
- [14] D. Benedettelli. *Squadre di robot mobili basati su tecnologia LEGO Mindstorms*. PhD thesis, Università degli Studi di Siena. <http://anderslemke.dk/multiagent/report.pdf>.
- [15] D. Benedettelli. LEGO Rubik Utopy - NXT Rubik’s cube solver, 2007. <http://daniele.benedettelli.com/LRU.htm>.

- [16] D. Benedettelli. *Programming LEGO NXT robots using NXC (beta 30 or higher - version 2.2)*, June 2007.
- [17] W. Cazzola. Linda. Technical report, Università degli Studi di Milano.
- [18] R. Coyne, R. Lucas, J. Li, M. Parker, and J. Lee. *Co-operation and complicity - Voices, robots and tricksters in the digital marketplace*. PhD thesis, University of Edimburgh.
- [19] A. De Rosi and C. Dominijanni. Progetto del Corso di Intelligenza Artificiale. Technical report, Università ‘Roma Tre’, 2000.
- [20] J. D.Miller, M. R.Hoehn, and D. J.Campbell. *Modelling X-Ray diffraction with LEGO NXT System*. PhD thesis, Bradley University.
- [21] F. Ferrazza. Nuove tecnologie - robot da allevamento. *Galileo*, Gennaio 2006.
- [22] M. Garbati. Complessità e comportamenti emergenti. il contributo della robotica educativa. *Le Scienze - Web News*.
- [23] M. Garbati. Robotica educativa. *Le Scienze - Web News*.
- [24] J. Hansen. *Not eXactly C - Programmer’s guide(version 1.0.1 b33)*, October 2007.
- [25] S. Hassenplug. Technical report.
- [26] A. Lemke, J. Laidlaw, and L. Zilmer-Pedersen. *Developing Multi-Agent Lego Robotics*. PhD thesis, DTU, 2007.
- [27] F. Molfese. *Progetto e sviluppo di un sistema per la comunicazione wireless tra robot LEGO*. PhD thesis, Università degli Studi di Modena e Reggio Emilia, 2004.
- [28] E. Serra. *Realizzazione di sistemi di controllo prototipali mediante LEGO Mindstorms*. PhD thesis, Università degli Studi dell’Aquila, Dipartimento di Ingegneria Elettrica e dell’Informazione.
- [29] Silberschatz, Galvin, and Gagne. *Operating System Concepts with Java*. Wiley, 6th edition, 2004.
- [30] W. Spinelli. Comunicazione e cooperazione di robot Lego. Technical report, 2001.
- [31] S. Toledo. Technical report.
- [32] University of Aarhus, Department of Computer Science. *RCX Manual*, 2006.
<http://legolab.daimi.au.dk/CSaEA/RCX/Manual.dir/RCXManual.html>.