Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea Triennale in Ingegneria Informatica

TESI DI LAUREA

Interfacciamento di URBI con robot didattico Lego Mindstorms NXT

Candidato:
Michielan Marco
Matricola 522251

Relatore:
Ch.mo Prof.Michele Moro

Anno Accademico 2008-2009

Ringraziamenti

Desidero innanzitutto ringraziare il Professor Michele Moro per avermi seguito durante lo svolgimento di questo elaborato e per la grande disponibilità e cortesia dimostratemi.

Inoltre desidero ringraziare il Professor Emanuele Menegatti per le idee ed i consigli fornitemi, la ricercatrice Dottoressa Monica Reggiani per avermi aiutato nelle operazioni di compilazione e il Dottor Stefano Ghindoni per il supporto fornitomi in laboratorio.

Un ringraziamento è dovuto a tutti i docenti del corso di laurea triennale in Ingegneria Informatica.

Infine desidero ringraziare i miei genitori e mio fratello per aver creduto in me e per non avermi mai fatto mancare il loro sostegno morale ed economico durante questo mio percorso di studi, anche nei momenti di grande difficoltà.

Sommario

Obiettivo Generale

Interfacciare URBI con il robot Lego Mindstorm NXT e il robot WowWee Roboreptile per dimostrare la versatilità, la flessibilità e la portabilità di tale piattaforma.

Contesto Operativo

I sistemi hardware utilizzati in questo elaborato sono stati un notebook Acer Aspire 6930 ed un pc del laboratorio di Robotica Autonoma (DEI-O).

I sistemi operativi coinvolti in questo progetto sono stati: Microsoft Windows Vista , Microsoft Windows XP e Ubuntu 9.4 (linux).

Il software della quale ci si è avvalsi è invece: URBI Engine for Lego Mindstorm NXT, Dev-C++, Code::Blocks, Mingw32 versione 5.1.4, urbi-sdk-2.0 e Visual Studio 2008.

Strumenti Utilizzati

Sono stati utilizzati un robot Lego Mindstorm NXT con tutti i sensori disponibili, un robot WowWee Roboreptile ed una torre infrarossi Lego collegabile al PC trammite porta seriale.

Risultati Raggiunti

Si è riusciti ad interfacciare completamente il robot Lego Mindstorm NXT ed il robot WowWee Roboreptile con la piattaforma URBI dimostrandone la flessibilità.

Indice

1 Introduzione **1**

2 Descrizione del robot Lego Mindstorms NXT **5**
2.1 Specifiche Tecniche 5
2.2 Kit di sviluppo 7
2.3 Programmazione 7
2.4 Sensori 9
2.5 Servomotori 14

3 Presentazione della piattaforma universale URBI **17**
3.1 Sintassi Generale del linguaggio urbiScript 18

4 Descrizione del robot WowWee Roboreptile **27**

5 Interfacciamento del NXT con il linguaggio URBI **31**
5.1 Presentazione della distribuzione di URBI per il controllo del robot Lego Mindstorms NXT 31
5.1.1 Motori 32
5.1.2 Sensori 34
5.1.3 Batteria e Altoparlante 35
5.1.4 Altre funzionalità di URBI for Lego Mindstorms NXT 35
5.2 Interfacciamento dei sensori non supportati dal linguaggio URBI 37
5.2.1 Sensori non supportati inizialmente 37
5.2.2 Il bus I^2C e il suo protocollo di comunicazione 37
5.2.3 Il protocollo dei comandi diretti del robot didattico Lego Mindstorms NXT 38
5.2.4 Codice URBI per l'interfacciamento dei sensori non supportati inizialmente 49

6 Dimostrazione di stabilità del sistema URBI - NXT **57**
6.1 Disabilitazione del PID dei servomotori 57

6.2 Esperimento 61

7 Interfacciamento del robot WowWee Roboreptile con la piattaforma URBI 65

7.1 Introduzione 65
7.2 Codice per l'interfacciamento del robot con URBI 67
7.3 Esperimento: Uso del linguaggio URBI con i robot NXT e Roboreptile 82

Manuale Utente 87

Manuale Tecnico 91

Conclusioni 99

Bibliografia 101

Capitolo 1

Introduzione

Il robot didattico **Lego Mindstorms NXT** può essere programmato tramite svariati linguaggi di programmazione. Generalmente il robot viene programmato con il linguaggio di programmazione NXT-G basato su un'interfaccia grafica ad icone. Si tratta quindi di un linguaggio di programmazione alquanto semplice ma un pò limitato. Per questo motivo sono stati sviluppati altri linguaggi per la programmazione di tale robot. I principali sono:

- **NBC:** un linguaggio assembly per la programmazione del brick NXT.
- **NXC:** un linguaggio di programmazione per NXT simile al C.
- **Lejos:** un firmware da sostituire a quello originale per poter programmare il robot NXT con il linguaggio Java.
- **URBI:** una piattaforma universale per il controllo dei robot.

Questa trattazione ha l'obiettivo di analizzare e sperimentare il funzionamento di **URBI** con il robot NXT ed anche con altri robot.

Per osservare il comportamento di URBI, dopo aver studiato la documentazione disponibile online all'indirizzo www.gostai.com/doc.php e aver scaricato la piattaforma **URBI for Lego Mindstorms NXT** da www.gostai.com/lego.html compilando l'apposito form, ho provato il funzionamento del robot Mindstorm NXT interfacciato con URBI. Una volta testato il suo uso, sono passato allo sviluppo delle interfacce per l'impiego dei sensori non supportati inizialmente da URBI. Infatti la piattaforma base permette solamente di sfruttare i sensori contenuti nel kit del robot, cioè:

- sensore di pressione (touch sensor)
- sensore di luce (light sensor)

- sensore di suoni (sound sensor)
- sensore ultrasuoni / di distanza (ultrasonic sensor)

Gli altri sensori che il dipartimento, nel laboratorio GIRST, mette a disposizione, ma che non sono supportati dalla distribuzione di URBI per Lego Mindstorm NXT sono:

- un sensore di accellerazione/accellerometro (Acceleration-Tilt sensor)
- un sensore di colori (Color sensor)
- una bussola (Compass sensor)
- un giroscopio (Gyro sensor)
- un rilevatore di infrarossi (IRSeeker sensor)
- un sensore di comunicazioni infrarossi (IRLink sensor)

I sensori appena elencati, ad eccezione del giroscopio, sono tutti sensori digitali, cioè usano il bus I^2C. Dopo aver analizzato il protocollo per comunicare con questi sensori, ho sviluppato il codice che ne consente l'utilizzo.

Ultimate le prove dei sensori, sono passato ad un analisi del sistema complessivo URBI-NXT dal punto di vista della stabilità e dei ritardi che la piattaforma comporta. Per questo ho effettuato un esperimento nel quale ho analizzato il comportamento del robot interfacciato con URBI.

Una seconda parte dell'elaborato si è concentrata sulla flessibilità della piattaforma URBI: ho provato l'interfacciamento di un secondo robot, il Roboreptile della WowWee. Il Reoboreptile è un robot giocattolo comandato trammite segnali infrarossi inviati da un apposito telecomando. Per questo motivo è stato necessario documentarmi in internet per trovare il protocollo dei comandi inviati dal telecomando al robot giocattolo. Queste informazioni le ho trovate in un sito amatoriale all'indirizzo
http://evosapien.com/robosapien-hack/nocturnal/RoboReptile/. Sempre durante questa attività di ricerca ho trovato un applicativo ms-dos che permette di inviare segnali infrarossi attraverso la torre infrarossi della Lego. Questo dispositivo veniva usato per programmare il precedecessore del robot Lego Mindstorm NXT chiamato Lego RCX. L'applicativo che permette l'invio di segnali trammite tale dispostitivo è disponibile nel sito
http://www.robotika.sk/mains.php?page=/projects/robsapien/ nel quale si discute su come controllare altri robot della ditta WowWee, ed in particolare su come pilotare il Robosapien. Dopo aver testato il funzionamento di tale applicativo con il Roboreptile, sono passato allo sviluppo di un plug-in per

URBI che permetta la gestione delle segnalazioni infrarossi verso il robot. Per ottenere il plug-in ho dovuto studiare la documentazione di URBI per la creazione di nuovi oggetti da caricare nell'engine della piattaforma. Questi oggetti vengono detti UObject. Dopo aver sviluppato il codice del plug-in, lo ho compilato: ho scelto di compilarlo sia in ambiente Linux, nella quale non ho riscontrato problemi, sia in ambiente Microsoft Windows nella quale la compilazione non ha avuto esito positivo. Dopo essere riuscito a compilare il codice in Linux, ho provato il robot, ottenendo il pieno controllo di tale dispositivo con risultati eccellenti: il robot si comporta proprio come se fosse comandato dal apposito telecomando. Per dimostrare la flessibilità della piattaforma ho creato un layout di comandi per il robot NXT che corrisponda ai comandi del robot Roboreptile ed ho provato a far eseguire le stesse operazioni ai due robot. Con questa prova ho notato che il robot NXT è molto più agile e veloce dal Roboreptile ma il comportamento finale dei due robot è uguale. Di conseguenza sono riuscito a dimostrare la flessibilità della piattaforma, dato che ho lavorato su più robot e su sistemi operativi diversi.

Capitolo 2

Descrizione del robot Lego Mindstorms NXT

Il robot Lego Mindstorms NXT è un robot didattico sviluppato e rilasciato dalla Lego nel Luglio del 2006. Il componente principale del kit è il computer chiamato NXT Brick.

Figura 2.1: Brick NXT

2.1 Specifiche Tecniche

Esso è dotato di:

- microprocessore centrale ARM7TDMI a 48 MHz e 32-bit. ARM7TDMI è una CPU RISC progettata dalla ARM, basata sull'architettura ARM v4T ed è stata studiata per dispositivi mobili e a bassa potenza. Questo

processore supporta istruzioni a 32-bit e a 16-bit attraverso, rispettivamente, i set di istruzioni ARM e Thumb. Inoltre il microprocessore è supportato da 256 KBytes di memoria flash e 64 Kbytes di memoria RAM.

- microcontroller (PIC) a 8 bit ATmega48 a 4 MHz con 4 KByte di memoria flash e 512 Bytes di RAM. ATmega48 è un processore della famiglia di processori RISC AtmelAVR. La sua caratteristica principale è quella di utilizzare una memoria flash interna per memorizzare il contenuto del programma: questo permette di cancellare la memoria di programma e riscriverla con una nuova versione in pochi secondi e anche senza rimuovere il microcontrollore dalla scheda su cui è montato, velocizzando enormemente il processo di correzione e messa a punto del codice. Questa memoria di programma flash può essere riscritta circa 1000 volte.
- CSR BlueCore 4 Bluetooth controller che lavora alla frequenza di 26 MHz. Il dispositivo bluetooth ha memoria flash esterna da 8 Mbit e una memoria RAM da 47 KBytes.
- un display LCD con matrice 100x64 pixel
- una porta USB 1.1 full speed (velocita di trasmissione 12 Mbit/s)
- 4 porte di input con piattaforma digitale a 6 fili alle quali è possibile collegare i sensori o altri dispositivi di input.
- 3 porte di output con piattaforma digitale a 6 fili alle quali è possibile collegare i motori o altri dispositivi di output

Il brick NXT include un firmware open source rilasciato dalla Lego che supporta una macchina virtuale secondo le tecnologie Labview di National Instruments.
Il kit aducativo del robot Lego Mindstorms NXT include:

- il brick NXT
- 431 parti Lego Technic
- 3 servomotori
- un sensore ad ultrasuoni (Ultrasonic Sensor) per misurare le distanze
- un sensore di suoni (Sound Sensor) che misura il livello sonoro in base a dei modelli sonori preimpostati

- un sensore di luce (Light Sensor) che misura l'intensità della luce
- un sensore di contatto (Touch Sensor) che rileva la sua pressione, il suo rilaascio o le collisioni

Inoltre si possono acquistare dei sensori supplemetari. La HiTechnic, ad esempio, fornisce questi sensori compatibili:

- un sensore di accellerazione/accellerometro (Acceleration-Tilt sensor)
- un sensore di colori (Color Sensor)
- una bussola (Compass Sensor)
- un giroscopio (Gyro Sensor)
- un rilevatore di infrarossi (IRSeeker Sensor)
- un sensore di comunicazioni infrarossi (IRLink Sensor)

2.2 Kit di sviluppo

La Lego ha, inoltre, rilasciato:

- un Software Development Kit (SDK) che include informazioni sugli host USB driver, formato dei file eseguibili e informazioni sul bytecode per gli sviluppatori di software
- un Hardware Development Kit (HDK) che include documentazione e schemi elettrici per NXT brick e i sensori per gli sviluppatori di hardware
- un Bluetooth Development Kit (BDK) che documenta i protocollli usati per la comunicazione via Bluetooth

2.3 Programmazione

Alcuni programmmi molto sempilici possono essere scritti usando il menù del NXT mentre i programmi veri e propri possono essere caricati sul brick o via USB o via Bluetooth e possono essere scritti con vari linguaggi di programmazione:

- NXT-G : linguaggio di programmazione visuale di tipo drag and drop che si trova incluso nel kit educativo Lego Mindstorms NXT. Anche questo software è basato su LabVIEW di National Instruments ed è un applicativo semplice da usare, in modo che possa essere usato dai bambini. Questo strumento mette a disposizione i blocchi per tutte le principali funzioni e, questi blocchi possono essere personalizzati. Con un po di esperienza si può ottentere un livello di programmazione avanzata.

- NBC : (Next Byte Codes) un linguaggio di programmazione open-source a livello assembly. Il robot NXT ha un interprete dei comandi che può essere usato per eseguire programmi. Il compilatore NBC traduce il programma sorgente nel byte-code del NXT, che può essere eseguito direttamente dal brick NXT.Il linguaggio NBC descrive la sintassi assembly con la quale scrivere i programmi mentre la NBC Application Programming Interface (API) descrive le funzioni , le costanti e le macro che possono essere utilizzate dai programmi. Queste API sono raccalte in un header file che deve essere importato all'inizio del programma per poter utilizzrle.

- NXC : (Not eXactly C) un linguaggio di programmazione open source con sintassi simile al C. Il compilatore NXC non fa altro che tradurre il codice NXC in NBC è poi richiamare il compilatore di NBC.

- RobotC : un potente linguaggio di programmazione basato sul C, con un abiente Windows per scrivere e fare il debug di programmi.

- Lejos NXJ : è un linguaggio di alto livello open source basato sul linguaggio di programmazione Java che utilizza un firmware ad hoc sviluppato dal team Lejos. Questo firmware è una macchina virtuale java alleggerita che è stata portata sul brick NXT.
  Lejos NXJ offre quindi un linguaggio orientato agli oggetti, preempitve thread, array multdimensionali, ricorsività, sincronizzazione, eccezzioni, tutti i tipi supportati da Java (float, long e String) molte delle librerie come java.lang, java.util e java.io ed una documentazione approfondita di tutte le API.

- Lejos OseK: una piattaforma open source per programmare il robot in C/C++ con un sistema operativo real-time che include le specifiche di porting per il processore ARM7.Quindi questa piattaforma può girare anche sul robot NXT poichè tale robot possiede il processore i questione.

- URBI : la piattaforma universale per il controllo dei robot che descriveremo in questa trattazione nel capitolo seguente.

- MatLab : un linguaggio di programmazione di alto livello per elaborazioni numeriche, raccolta dati e analisi. Può essere utilizzato per controllare il Lego NXT via Bluetooth o via USB, usando un open source chiamato RWTH - Mindstorms NXT Toolbox. Questo toolbox utilizza il protocollo "LEGO MINDSTORMS NXT Bluetooth Communication Protocol" per comandare il brick.

- Simulink : è un ambiente basato su MatLab per modellare e simulare sistemi dinamici. Usando questo ambiente si possono progettare algoritmi di controllo, produrre il relativo codice C e caricare il codice compilato sul NXT.

- Lua : un linguaggio di scripting per il controllo del robot NXT.

- Ruby-NXT : è una libreria per programmare il robot NXT con il linguaggio di programmazione Ruby. A differenza degli altri linguaggi di programmazione che inviano il codice già compilato al robot, questo linguaggio invia direttamente i comandi via bluetooth o usb utilizzando il protocollo dei comandi diretti (Direct Command Protocol).

2.4 Sensori

Il sensore ad ultrasuoni (Ultrasonic Sensor) permette al NXT di stimare le distanze e vedere dove si trovano gli oggetti attorno a lui. Questo sensore può misurare distanze in centimetri e in pollici. La massima distanza misurabile è di 255 cm con una precisione di circa 3 cm. Per ottenere la stima della distanza, il sensore invia dei suoni ultrasonici e misura quanto tempo intercorre perchè questi, una volta che hanno raggiunto l'oggetto davanti al robot, rimbalziono e ritornino al sensore. E per questo che il sensore è più sensibile e preciso con grandi oggetti e superfici piane mentre può anche non funzionare correttamente con oggetti sottili, piccoli e superfici non piane come cilindri, sfere di piccolo raggio. Questo sensore è di tipo digitale, usa quindi il bus I^2C.

Figura 2.2: Ultrasonic Sensor

Il sensore di suoni (Suond Sensor) invece è un sensore analogico che fornisce un valore proporzionale al suono rilevato. Il sensore rileva suoni fino a circa 90 dB. Ci sono due modalità in cui il sensore lavora: dB misura i suoni in decibel e dBA misura i suoni in Adjusted decibel cioè quelli udibili all'orecchio umano. Essendo molto complicatro rilevare una valore assoluto, la misurazione viene indicata in percentuale con una distriubuzione come la seguente: dal 4% al 5% per i rumori di sottofondo, dal 5% al 10% per una persona in lontananza che parla, dal 10% al 30% per una normale conversazione tenuta vicino al sensore e dal 30% al 100% per persone che urlano o musica ad alto volume.

Figura 2.3: Sound Sensor

Il sensore di luce (Light Sensor) è un sensore analogico che fornisce un valore proporzionale alla luminosità rilevata nell'ambiente o alla luce riflessa dell'emettitore. Per questo motivo si possono settare i due modi di funzionamento: ambient e reflected. Questo sensore permette inoltre di distinguere le varie tonalità di grigio. Anche questo sensore restituisce un valore in percentuale.

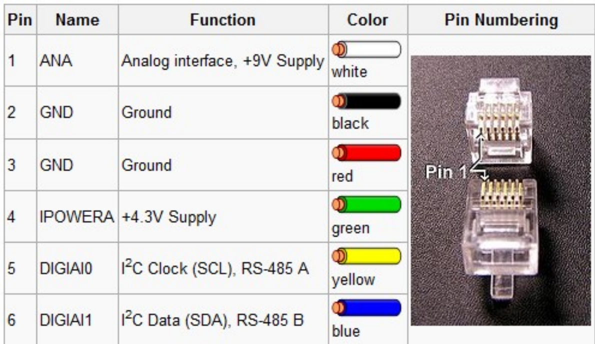
Figura 2.4: Light Sensor

Il sensore di contatto (Touch Sensor) è un sensore booleano che indica se il pulsante è premuto (ON,1) o rilasciato (OFF,0). E' inoltre possibile settare il sensore per ottenere il valore RAW che va da 0 a 1023.

Figura 2.5: Touch Sensor

I sensori sono connessi con gli appositi cavi con connettori a 6 posizioni che caratterizzano sia l'interfaccia analogica che quella digitale. Si riporta lo schema di collegamneto dei conettori.

Figura 2.6: Schema dei cavi e dei connettori per i sensori del NXT



Pin	Name	Function	Color	Pin Numbering
1	ANA	Analog interface, +9V Supply	white	Pin 1
2	GND	Ground	black	
3	GND	Ground	red	
4	IPOWERA	+4.3V Supply	green	
5	DIGIAI0	I^2C Clock (SCL), RS-485 A	yellow	
6	DIGIAI1	I^2C Data (SDA), RS-485 B	blue	

Si descrivono ora i sensori opzionali e le loro caratteristiche fondamentali: Il sensore accellerometro (Acceleration / Tilt Sensor) è un sensore digitale che rileva le accelerazioni lungo i tre assi cartesiano x, y e z in u range di -2g e 2g (g è l'accelerazione gravitazionale terrestre e vale 9,80665 m/s^2) scalati di 200 valori per g.

Figura 2.7: Accelerometer - Tilt Sensor

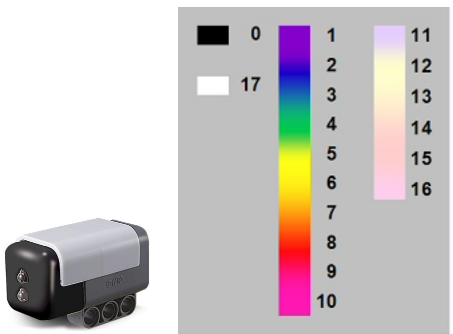
Per leggere i valori di questo sensore è necessario accedere ai registri descritti nella figura seguente:

Figura 2.8: Registri del accelerometro

Address	Type	Contents
42H	byte	*X* axis upper 8 bits
43H	byte	*Y* axis upper 8 bits
44H	byte	*Z* axis upper 8 bits
45H	byte	*X* axis lower 2 bits
46H	byte	*Y* axis lower 2 bits
47H	byte	*Z* axis lower 2 bits

Il sensore di colori (Color Sensor) è un sensore digitale che rileva le colorazioni degli oggetti in una gamma di 17 colori riportati nella seguente immagine.

Figura 2.9: Color Sensor e ColorNumber



Per conoscere i valori rilevati dal sensore si deve accedere ai registri qui sotto riportati.

Il sensore Bussola (Compass Sensor) è un sensore digitale che rileva la posizione in gradi rispetto al nord. Il nord è quindi rappresentato dal valore 0, l'est da 45, il sud da 90 e l'ovest dal 135.

Il sensore giroscopico (Gyro Sensor) è un sensore analogico che rileva le accelerazioni lungo un solo asse (quello orizontale se il sensore viene utilizzato

Figura 2.10: Registri del Sensore di colori

Address	Type	Contents
00 – 07H	chars	Sensor version number
08 – 0FH	chars	Manufacturer
10 – 17H	chars	Sensor type
18 – 3DH	bytes	Not used
3E, 3FH	chars	Reserved
40H	byte	Not used
41H	byte	Command
42H	byte	Color number
43H	byte	Red reading
44H	byte	Green reading
45H	byte	Blue reading

Figura 2.11: Compass Sensor

in posizione normale) e restituisce il numero di gradi per secondo che rileva, indicando anche il senso di rotazione.

Figura 2.12: Gyro Sensor

Il ”sensore” IRLink permette al NXT di controllare altri dispositivi come ad esempio i treni Lego. Questo sensore permette inoltre al NXT di comunicare con il Lego RCX (suo predecessore) leggendo valori dai sensori e controllando i motori.

Figura 2.13: IRLink Sensor

Il ”sensore” IRSeeker permette invece al NXT di percepire fonti di segnali infrarossi con un angolazione di 240 gradi. E’ possibile rilevare un segnale infrarosso e farlo seguire dal robot, per questo la Hitechnic ha creato una palla che emette segnali infrarossi in tutte le direzioni per rendere possibile il gioco del calcio agli NXT.

Figura 2.14: IRSeeker Sensor

2.5 Servomotori

I tre servomotori funzionano in corrente continua (DC) e hanno le seguenti caratteristiche:

- Tensione di alimentazione 9V (DC)
- Velocità massima 170 rpm (117 rpm a 9V)
- Potenza meccanica a 9V 2,03W
- Potenza elettrica a 9V 4,95W
- Efficienza a 9V 41%
- Assorbimento a 9V 0.55A
- Corrente di No-Load 60mA
- Coppia a 9V 16,7 N*cm

- Coppia in stallo 50 N*cm
- Corrente di stallo 2A
- Peso 80 gr

Inoltre ogni servomotore ha al suo interno un sensore di rotazione che permette di misurare la velocità e/o distanza percorsa da un certo punto.

Figura 2.15: Servomotore del NXT

Il robot Lego Mindstorms NXT può essere montato a piacere, anche se esistono dei modelli standard. Questi modelli standard sono:

- Alpha Rex

- Spike

- RoboArm T-56

- TriBot

In questa trattazione ho utilizzato il modello Tribot semplificato ottenuto non montando le pinze anteriori e il sensore di luminosità ed usando solamente il sensore ultrasuoni.

Capitolo 3

Presentazione della piattaforma universale URBI

URBI è una piattaforma universale per il controllo dei robot. Il nome URBI è l'acronimo di ***Universal Real-Time Behavior Interface***. I quattro requisiti della piattaforma universale sono:

- **Flessibilità:** può essere usato con un qualsiasi robot, con ogni sistema operativo e con qualunqe linguaggio di programmazione
- **Modularità:** incorpora un architettura a componenti modulari. Questi componenti sono trasparentemente inglobati o nella piattaforma o nel computer remoto.
- **Potenza:** nel campo della robotica e dell'intelligenza artificiale sono necessarie astrazioni potenti della programmazione ad eventi e del parallelismo.
- **Semplicità:** nel padroneggiare tale piattaforma non dovrebbe essere necessario leggere centinaia di pagine di documentazione in modo da renderla accessibile anche a persone poco esperte come bambini e hobbisti, per stimolare la creatività a beneficio della robotica.

URBI è un *middleware* che include un architettura a componenti chiamati **UObject**, un linguaggio di scripting chiamato **urbiScript** e alcuni strumenti di programmazione grafica raccolti in **urbiStudio**.

Il cuore di questa tecnologia è basato su **urbiScript**, un nuovo linguaggio di scripting che porta caratteristiche innovative in termini di parallelismo, programmazione basata sugli eventi e oggetti distribuiti. La sintassi di questo linguaggio di scripting è molto simile a quella del C++ in modo da renderla semplice da imparare. Urbiscript è un linguaggio di programmazione orientato agli oggetti, basato su un approccio a prototipi.

Con questa piattaforma è possibile incapsulare il linguaggio di scripting urbiScript in un codice scritto con un linguaggio di programmazione tra i più conosciuti come ad esempio Java, MatLab. Per far questo ci si avvale di apposite librerie da impotare nel linguaggio di programmazione scelto:le **LibUrbi**. Con queste librerie è possibile:

- aprire una connessione con il robot
- inviare comandi al robot
- richiedere e ricevere il valore di una variabile del robot
- ricevere i messaggi inviati dal robot e reagire adeguatamente

il tutto direttamente dal linguaggio. Per questo nell'acronimo compare il sostantivo *Interface*.
Un'altra potenzialità di URBI è la possibilita di programmare il robot come se fosse una macchina a stati finiti. Gli stati sono chiamati Behavior. Per ogni stato si crea una funzione che rappresenta il comportamento di quello stato e, per ogni transizione si attivano e disattivano (tramite comando stop) gli stati interessati. Per questo motivo nell'acronimo compare il sostantivo *Behavior*.

Questa piattaforma permette di importare degli oggetti C++ ed inserirli in urbiScript per usarli come un oggetto presente nativamente nel linguaggio. Si può inoltre creare un UObject come un *oggetto remoto*, un semplice eseguibile autonomo per Windows, Linux e Mac OSX. Oltre tutto questo è possibile passare un UObject da embedded a remoto senza dover cambiare alcuna riga di codice.

3.1 Sintassi Generale del linguaggio urbiScript

Come tutti i linguaggi di programmazione, il linguaggio urbiScript prevede l'utilizzo di variabili definibili nel modo seguente:

```
var x = 0 ;
var y ;
```

Mentre per leggere il valore della variabile è necessario solamente digitare:

```
x;
```

In URBI non c'è un concetto di variabili globali e locali ma si può gestire questa cosa mettendo un prefisso alle variabili per farle diventare "globali".

```
var myPrefix.x = 0 ;
```

In urbiScript sono definite anche le liste, e queste possono essere utilizzate nei modi seguenti:

```
mylist = [1,2,35.12,"hello"];
mylist;
[139464:notag] [1.000000,2.000000,35.120000,"hello"]
```

L'ultima riga rappresenta la risposta del server al comando presente nella seconda riga, che chiede di conoscere i valori contenuti nella lista mylist. Per aggiungere elementi ad una lista si usa l'operatore di concatenazione rappresentato dal simbolo +:

```
mylist = [1,2] + "hello";
mylist;
[146711:notag] [1.000000,2.000000,"hello"]
```

Si possono inoltre annidare liste in altre liste. Eccone un esempio:

```
x = 1;
mylist + [45,x];
[148991:notag] [1.000000,2.000000,"hello"
    ,[45.000000,1.000000]]
```

Nelle liste, come in gran parte dei linguaggi di programmazione, si può accedere ad un singolo elemento in base alla sua posizione. In urbiScript gli elementi sono numerati da 0 a n-1 dove n è il numero di elementi che la lista contiene. Per ottenere il valore di un elemento è necessario scrivere:

```
mylist[2];
[146711:notag] "hello"
```

Se si usano liste annidate, per esempio per ottenere delle matrici, si possono usare indici multipli. Vediamo un esempio:

```
mylist[3][1];
[146712:notag] 45.000000
```

Le liste in URBI prevedono inoltre due metodi, head e tail che restituiscono rispettivamente il primo elemento della lista ed il resto della lista, escluso il primo elemento. Per esempio:

```
mylist = [1,2,"hello"];
head(mylist);
[146711:notag] 1.000000
tail(mylist);
[146711:notag] [2.000000,"hello"]
```

Il linguaggio urbiScript prevede l'esecuzione parallela di due comandi. Il separatore che ha il compito di segnalere questo fatto è "&". Ad esempio,

nel codice seguente,la ”&” forza i due comandi ad iniziare contemporaneamente. Questo significa che un comando non può iniziare se l'altro non è completamente avviabile.

```
x=4 time:1s & y=2 speed:0.1;
```

In opposizione ai comandi eseguiti in parallelo si trovano i comandi eseguiti strettamente in modo seriale. Per far questo è necessario apporre come separatore il simbolo ”|”. In questo modo si segnala che il secondo comando deve iniziare esattamente quando finisce il primo. Proprio per questo motivo il primo comando attende che il secondo sia completamente avviabile prima di essere eseguito.

```
x=4 time:1s | y=2 speed:0.1;
```

Il separatore ”,” invece, indica al server di eseguire i due comandi separatamente ma di far partire il secondo il prima possibile senza imporre alcuna restrizione. Infine il separatore ”;” segnala al sever che il comando, la serie o il parallelo di comadi sono terminati.

Eseguendo due o più comandi in parallelo è possibile che avvengano dei conflitti di assegnazione alle varibili, come nel seguente esempio.

```
x=1 & x=5;
```

In questi casi è possibile indicare al server sul come comportarsi con il comando blend.

```
x->blend = add;
```

Ci sono numerose modalità di blending che sono riportate nella tabella seguente:

- add: somma i due valori numerici in conflitto
- mix mode: restituisce la media dei valori in conflitto
- queue: restituisce il valore dell'ultima assegnazione che ha creato il conflitto
- discard: restituisce il valore precedente all'ultima assegnazione che ha creato il conflitto
- cancel: rende vuota/nulla la variabile in conflitto

Il linguaggi urbiScript prevede alcuni comandi utili per l'utilizzo della piattaforma. Questi comandi sono descritti nell'elenco seguente:

- reset: esegue un riavvio virtuale del robot, cancellando tutti gli script in esecuzione in quel momento

- stopall: ferma l'esecuzione di tutti i comandi in ogni connessione. Utile quando si perde il controllo del server a causa di script in loop infinito
- reboot: esegue il riavvio fisico del robot
- shutdown: ferma il robot
- uservars: restituisce la lista delle variabili usate

I costrutti di controllo if, while, for sono uguali a quelli del C/C++ mentre ci sono dei costrutti specifici per la gestione degli eventi:

- il costrutto **at** serve a catturare un evento. Per esempio, nel codice seguente, la scritta "Obstacle appears" comparirà solamente quando la variabile distance avrà un valore minore di 50.

  ```
          at (distance < 50)
  echo "Obstacle appears";
  ```

  Questo comando è molto utile per fare in modo che il robot reagisca prontamente agli eventi.

- il costrutto **onleave** si comporta come l'else per il costrutto if, cioè rappresenta il comando alternativo nel caso in cui l'evento non si verifichi. Vediamo un esempio:

  ```
  at (distance < 50)
  echo "Obstacle appears"
  onleave
  echo "The obstacle is gone";
  ```

- il costrutto **whenever** si comporta in modo simile al costrutto while e si può descrivere con la frase seguente: finché non avviene un evento ripeti una certa serie di operazioni. Per esempio, il codice successivo fa scrivere a video "Obstacle appears" finché la variabile distance è minore di 50.

  ```
  whenever (distance < 50)
  echo "There is an obstacle";
  else
  echo "There is no obstacle";
  ```

  Come si può notare, si può usare il costrutto **else** per indicare cosa si deve fare quando l'evento non è più verificato.

- il costrutto **wait(n)** fa attendere il sistema per n millisecondi.

- il costrutto **waituntil(test)** fa attendere il sistema fino a che non si verifica la condizione test. Questo comando può essere utilizzato per sincronizzare programmi differenti in parallelo.
- il costrutto **timeout(n) cmd** fa eseguire il comando cmd e lo ferma se non è ancora terminato entro n millisecondi.
- il costrutto **stopif(test) cmd** fa eseguire il comando cmd e lo arresta non appena si verifica la condizione test. Se il comando è già terminato quando si verifica la condizione non succede nulla.
- il costrutto **freezeif(test) cmd** fa eseguire il comando cmd e lo congela non appena si verifica la condizione test.

Nel linguaggio urbiScript è inoltre possibile sollevare degli eventi e catturarli. Per sollevare un evento, per esempio, si deve digitare:

```
emit myevent(1,"hello");
```

Per catturare l'evento dell'esempio precedente si deve inviare il seguente pezzo di codice:

```
at (myevent(x,y))
  echo "catch two: " + x + " " + y;
```

Un altra caratteristica degli eventi è quella che si possono creare dei filtri che catturano delle condizioni particolari degli eventi. Per esempio, il codice seguente cattura solamente gli eventi che hanno come primo parametro 1:

```
at (myevent(1,x))
  echo "catch one: " + x;
```

Gli eventi hanno solitamente virtualmente una durata nulla anche se è possibile indicare la durata degli eventi come nell'esempio seguente.

```
emit(10s) boom;
emit(15h12m) myevent(1,"hello");
```

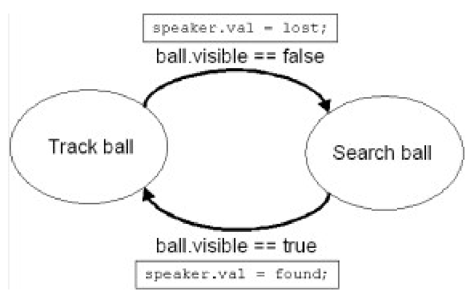
Con il linguaggio urbiScript si possono anche creare degli eventi ricorrenti, utillizzando il comando **every(time)**. Per esempio:

```
every (10m) echo "hello";
```

Come si può notare, questo comando solleva l'evento ogni 10 minuti all'infinito, ma a volte questo non va bene, allora per arrestare questo comando conviene sempre apporgli un tag e richiamare il comando stop nome-tag.

Un altra potenzialità del linguaggio di scripting urbiScript è la programmazione a stati finiti. Per rendere più chiaro questo concetto si riporta il grafico di una macchina a stati che implemeta l'inseguimento di una pallina ed il codice in urbiScript che traduce fedelmente il grafico.

Figura 3.1: Grafico degli stati del programma per l'inseguimento della pallina



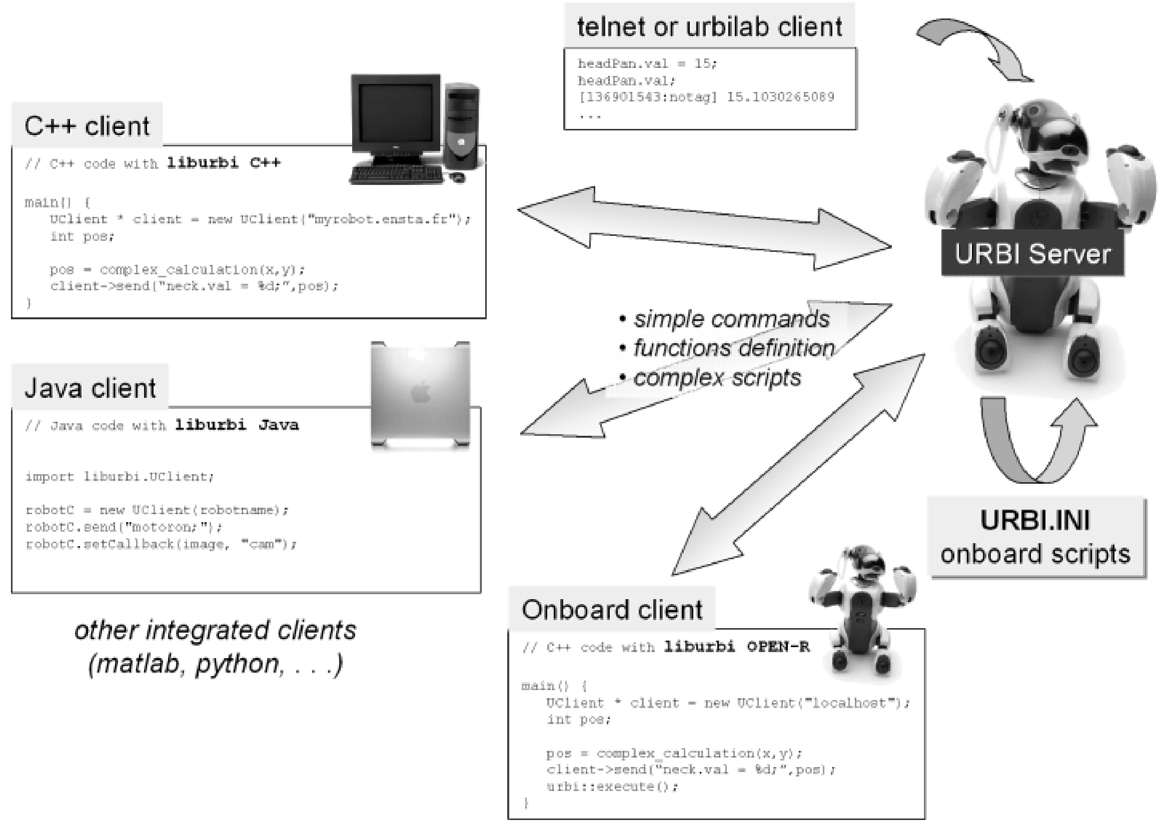

```
// Tracking state
function tracking() {
  whenever (ball.visible) {
    headPan  = headPan + ball.a * camera.xfov * ball.x
     &
    headTilt = headTilt+ ball.a * camera.yfov * ball.y;
  }
};

// Searching state
function searching() {
  period = 10s;
  {
    headPan'n  = 0.5 smooth:1s &
    headTilt'n = 1 smooth:1s
  } |
  {
    headPan'n  = 0.5 sin:period ampli:0.5 &
    headTilt'n = 0.5 cos:period ampli:0.5
  }
};
// Transitions
at (ball.visible ~ 100ms) {
  stop search;
  speaker = found;
  track: tracking();
};

at (!ball.visible ~ 100ms) {
  stop track;
```

```
  speaker = lost;
  search: searching();
};
```

Come si nota, gli stati vengono rappresentati in urbiScript come delle funzioni mentre le transizioni sono i comandi per la cattura degli eventi. Quando avviene un evento il programma cambia lo stato in cui si trovava precedentemente. In questo modo la programmazione dei comportamenti dei robot risulta più semplice. I comportamenti in inglese si dicono Behaviors.

In conclusione il linguaggio di programmazione urbiScript è molto simile al C ma ha delle caratteristiche particolari che lo rendono più adatto all'interfacciamento con molti dispositivi ed in particolare con una grande varietà di robot. Il grafico seguente illustra le connessioni e le impostazioni tipiche del architettura client e software di un applicazione URBI.

Figura 3.2: Architettura client e software di un applicazione URBI



Capitolo 4

Descrizione del robot WowWee Roboreptile

Il Roboreptile è un dinosauro robot dotato di sensori che rendono i suoi comportamenti e le sue reazioni quasi realistiche. Ogni suo movimento è accompagnato da un effetto sonoro. I sensori permettono al Roboreptile di percepire le cose che gli stanno intorno. Il Roboreptile è controllato da un apposito telecomando che controlla ogni suo movimento e/o comportamento. Il robot ha movimenti biomorfici ottenuti con meccanismi avanzati, può camminare su 4 e su 2 zampe, può saltare e simulare l'attacco.
Inoltre il Roboreptile è dotato dei sensori:

- un sensore di contatto (Touch sensor)
- un sensore stereofonico (Stereo sonic sensor)
- un vista ad infrarossi (Infrared vision)

che lo rendono interattivo con l'ambiente che lo circonda. Il Roboreptile possiede anche un intelligenza artificiale: reagisce agli stimoli esterni diversamente in base a quale dei tre stati d'animo è stato scelto. Gli stati d'animo disponibili sono:

- *affamato*: è quello stabilito di default, attivando il comando feed in questa modalità il robot segue il segnale lasciato dal telecomando. Con questo stato d'animo il roboreptile è aggressivo, esplora l'ambiente cercando la sua preda e quando la incontra la attacca ruggendo e mordendo.
- *soddisfatto*: il robot si calma dopo aver mangiato.
- *incappuciato*: il robot diventa sottomesso e a seconda dell'interazione umana, può svegliarsi affamato o cadere nel sonno.

Il telecomando del Roboreptile ha 3 layers: per ognuno di questi, ogni pulsante invia un comando diverso. I pulsanti sono 10. La tabella seguente riporta i comandi inviati dal controller.

Tabella 4.1: Comandi corrispondenti ai pulsanti in base ai 3 layer del Roboreptile

Pulsante	Layer 1	Layer 2	Layer 3
su	avanti	alzati su 2 zampe	salta
giù	indietro	ritorna su 4 zampe	perlustra
sinistra	girati a sinistra	abbassa il volume	colpo di coda a sinistra
destra	girati a destra	alza il volume	colpo di coda a destra
stop	stop	stop	stop
testa di sinistra	muovi la testa a sinistra	modalità programma	attacca
testa di destra	muovi la testa a destra	esegui programma	squotiti
demo	demo 1	demo 2	vertigini
roam	vagabonda	modalità guardia	morsica
feed	mangia	mangia	mangia

I segnali infrarossi inviati dal telecomando hanno una portante 39.2kHz e i dati vengono modulati con modulazione digitale binaria. Ogni segnale è lungo 12 bit ed è preceduto da un impulso di start della durata di ∼6.66 ms. Un uno viene codificato con una pausa di ∼3.33 ms, uno zero viene codificato con una pausa di ∼0.833 ms ed ogni pausa è seguita da un impulso di ∼0.833 ms per separare i bit. Questi segnali vengono rappresentati da numeri esadecimali. Nella tabella seguente riportiamo i segnali ralativi ai vari comandi rispetto ai tre layout.

Tabella 4.2: Segnali dei comandi 3 layer del Roboreptile

Comando	Layer 1	Layer 2	Layer 3
UP	0x481	0x491	0x4A1
DOWN	0x482	0x492	0x4A2
LEFT	0x483	0x493	0x4A3
RIGHT	0x484	0x494	0x4A4
STOP	0x485	0x495	0x4A5
HEAD LEFT	0x486	0x496	0x4A6
HEAD RIGHT	0x487	0x497	0x4A7
DEMO	0x488	0x498	0x4A8
ROAM	0x489	0x499	0x4A9
FEED	0x480	0x490	0x4A0

Capitolo 5

Interfacciamento del NXT con il linguaggio URBI

5.1 Presentazione della distribuzione di URBI per il controllo del robot Lego Mindstorms NXT

La Gostai ha sviluppato una particolare distribuzione di URBI nella quale l'engine contiene nativamente i comandi di interfacciamento con il robot Lego Mindstorms NXT. Questa distribuzione è scaricabile all'indirizzo http://www.gostai.com/lego.html. In questa pagina web si possono trovare anche dei tutorial e tutta la documentazione necessaria all'uso di tale piattaforma.

Urbi utilizza il protocollo dei comandi diretti per interfacciare l'NXT: di conseguenza il robot non riceve codice compilato ma il byte code dei comandi che gli vengono impartiti. Per sfruttare questa piattaforma si può collegare il robot o tramite il cavo USB in dotazione o trammite bluetooth. Se si usa il bluetooth è necessario far riconoscere il robot al PC ed usare i driver Lego Fantom Library altrimenti Urbi engine non riuscirà a comunicare con il robot.

La piattaforma ha un architettura di tipo client-server: il server ha il compito di comunicare con il robot, elaborando le informazioni e i comandi che il client invia al robot. Per utilizzarla è necesario far partire prima il server lanciando l'applicazione URBI NXT Server 1.5 Beta, dopo aver costruito il robot e installato la distribuzione in questione in ambiente Micrsoft Windows (provato in Vista ed in XP) con i relativi driver del NXT, e poi lanciando la console di comando URBI Console 1.5. Da questa applicazio-

ne si preme il pulsante sul quale c'è scritto "Connect to LEGO Mindstorms NXT" per iniziare la comunicazione con il robot. Se la connessione è riuscita lo sfondo del pulsante diventerà verde mentre se ci saranno errori diventerà rossa. Questa console permette all'utente di interagire con il robot NXT tramite il linguaggio di scripting urbiScript. Le modalità per editare il codice urbiScript sono due:

- direttamente inserendo le righe di codice nella casella di testo in fondo alla finestra della console e premendo il pulsante a destra *Send*; a mano a mano che si inviano delle righe di codice al sever, vengono compilate ed eseguite.

- scrivendo un file di testo con estensione .u e caricandolo dalla console premendo il pulsante *Select a file* e poi *Send*; il codice viene compilato e mandato in esecuzione riga dopo riga.

Figura 5.1: URBI Console 1.5

5.1.1 Motori

Nel linguaggio urbiScript, i tre servomotori del NXT sono rappresentati dagli oggetti wheelL, wheelR e claw e raggruppati in un unico oggetto wheels per comandarli tutti assieme. Ogni motore è un istanza dell'oggetto Servo. Un istanza di Servo è caratterizzata dai seguenti attributi:

- *val:* specifica la posizione del motore in gradi

- *speed:* specifica la velocità del motore nel range -100,100 (<0 il robot si muove all'indietro, >0 il robot si muove in avanti, 0 il robot si ferma)

- *PGain:* specifica il guadagno proporzionale (Proportional Gain) del PID del servomotore
- *IGain:* specifica il guadagno integrale (Integral Gain) del PID del servomotore
- *DGain:* specifica il guadagno derivativo (Derivative Gain) del PID del servomotore
- *Precision:* specifica la precisione del PID del servomotore
- *port:* specifica la porta alla quale il servomotore è connesso

Per ogni motore è possibile specificare la velocità cambiando l'attributo speed dell'oggetto che rappresenta quel motore. Per esempio per impostare la velocità del motore sinistro al 50% indietro basta digitare:

```
wheelL.speed=-50;
```

Per far muovere tutti i motori in avanti al 50% della velocità per 10 secondi basta editare il comando:

```
wheels.speed=50;wait(10s);wheels.speed=0;
```

Si può anche far accelerare e decelerare il robot impostando la velocità e il tempo che il robot deve impiegare per raggiungerla. Ad esempio se si vuole che il robot acceleri in 5 secondi per raggiungere il 50% della velocità, la mantenga per 30 secondi e poi impieghi 10 secondi per fermarsi si deve scrivere il seguente comando:

```
wheels.speed = 50 time:5s;wait(30s);wheels.speed = 0 time
    :10s;
```

I servomotori hanno al loro interno un sensore di rotazione (encoder) usato per rilevare la velocità di rotazione ed anche i giri effettuati dal motore da un certo momento. Per leggere la posizione, in gradi, in cui si trova un motore (in questo caso il motore destro) in un certo istante, basta inviare il comando:

```
wheelR.val;
```

che corrisponde al comando:

```
wheelR;
```

Si può inoltre far muovere il motore per un certo numero di gradi. Per esempio per far girare il motore destro di 280 gradi si deve digitare il comando:

```
wheelR.val = 280;
```

5.1.2 Sensori

Nel linguaggio di scripting urbiScript, ogni sensore del kit Lego Mindstorms NXT corrisponde ad un oggetto. Il sensore di luce è rappresentato dall'oggetto light, il sensore di contatto è rappresentato dall'oggetto bumper, il sensore di suoni è rappresentato dall'oggetto decibel ed il sensore ultrasuoni è rappresentato dall'oggetto sonar.
L'oggetto UltraSonicSensor ha i seguenti parametri impostabili:

- *val:* specifica la distanza misurata dal sensore in centimetri da 0 a 255
- *port:* specifica la porta sulla quale il sensore è connesso

L'oggetto sonar è un istanza di UltraSonicSensor. Per leggere il valore del sensore ultrasuoni basta inviare il comando:

```
sonar.val;
```

A questo comando il server risponde segnalando la distanza misurata dal sensore ultrasuoni in centimetri. Se la distanza misurata è superiore a 254 cm il server visualizza 255. Si possono impostare anche degli eventi legati ai sensori. Se, ad esempio, si vuole far arrestare il robot quando la distanza misurata dal sensore ultrasuoni è minore di 50 cm, lo si può fare inviando il seguente codice al server:

```
at (sonar.val < 50) wheels.speed = 0;
```

L'oggetto LightSensor ha i seguenti attributi:

- *val:* specifica il valore misurato dal sensore da 0 a 1
- *mode:* specifica il modo di funzionamento del sensore. I modi disponibili sono : `Ambiant` per misurazioni a luce ambiente, `Reflector` per misurazioni in ambienti chiusi per cui si rende neccesario illuminare la superficie con il led emettitore o `Normal` per misurazioni in cui si vuole ottenere il valore *raw*.
- *port:* specifica la porta sulla quale il sensore è connesso.

L'oggetto ligth è un istanza di LigthSensor.

L'oggetto Switch ha i seguenti attributi:

- *val:* specifica se il sensore è premuto o meno restiruendo corrispondentemente i valori 1 o 0.
- *port:* specifica la porte sulla quale il sensore è connesso.

L'oggetto bumper è un istanza di Switch.

L'oggetto SoundSensor ha i seguenti attributi:

- *val:* specifica il valore misurato dal sensore nell'intervallo 0-1.
- *mode:* specifica il modo di funzionamento del sensor: DB misura i suoni in decibel o DBA misura i suoni in decibel udibili.
- *port:* specifica la porta alla quale il sensore è connesso.

L'oggetto decibel è un istanza di SoundSensor.

5.1.3 Batteria e Altoparlante

Con urbiScript è possibile inoltre verificare lo stato della batteria rappresentata dall'oggetto Battery, ed utilizzare l'altoparlante interno al brick NXT tramite l'oggetto Beeper.
L'oggetto Battery ha il seguente attributo:

- *val:* specifica il livello di carica della batteria

L'oggetto Beeper mette a disposizione il seguente metodo:

- *play (frequency,duration):* fa suonare l'altoparlante del NXT alla frequenza frequency, i cui valori sono compresi tra 200 e 14000, per un intervallo di tempo duration in secondi. Se duration è uguale a zero l'altoparlante suona all'infinito.

Ad esempio per ottenere un beep a 200 Hz per 3 secondi si deve digitare:

```
beeper.play(200,3s);
```

5.1.4 Altre funzionalità di URBI for Lego Mindstorms NXT

Il linguaggio di scripting urbiScript permette di *applicare un tag ad un comando* per poter controllare la sua esecuzione. Ad esempio il comando *at* rimane in esecuzione indefinitamente sul server, quindi per poterlo fermare è neccessario indicizzarlo con un tag ed invocare il comando per arrestare un comando specificando il suo tag. Per eseguire un tag su un comando basta digitare il nome del tag e ':' prima del comando. Ad esempio:

```
mytag: at (decibel.val > 0.7) wheels.speed = 0;
```

Se si vuole che il robot non reagisca più all'evento del comando, si deve scrivere:

```
stop mytag;
```

Se non si specifica nessun tag, il tag di default è notag. Digitando

```
stop notag;
```

si arrestano tutti i comandi digitati precedentemente senza tag.

URBI permette anche di ottenere *movimenti ciclici a forma sinusoidale*. Ad esempio per ottenere che la velocità dei motori oscilli da -100 a 100 con un periodo di 10 secondi, si deve inviare il comando:

```
mytag:wheels.speed = 0 sin:10s ampli:100,
```

Oppure per fare in modo che la velocità dei motori oscilli da 0 a 100 con un periodo di 3 secondi, basta scrivere:

```
tag2:wheels.speed = 50 sin:3s ampli:50,
```

URBI gestisce il *Parallelismo* con molta semplicità. Infatti due comandi separati da "&" vengono eseguiti contemporaneamente ed iniziano esattamente nello stesso momento. Ad esempio:

```
wheelL.speed = 50 & wheelR.speed = -50;
```

Invece due comandi separati da ";" vengono eseguiti l'uno dopo l'altro nell'ordine in cui si trovano, come avviene in molti linguaggi di programmazione. Il separatore "," fa in modo che il primo comando venga eseguito in background mentre il secondo venga eseguito il prima possibile senza forzare l'avvio simultaneo dei due comandi.

In URBI si possono definire *funzioni*. Per esempio si riporta la funzione che muove il robot in avanti con una certa velocità e per un certo periodo di tempo:

```
function global.forward(speed,timer)
{
wheels.speed = speed;
wait(timer);
wheels.speed = 0;
},
```

Questa funzione può essere richiamata con il comando seguente:

```
global.forward(100,3000);
```

La console di URBI permette, come già detto in precedenza, di caricare sul server del codice scritto su un file di testo. Il file contentente il codice deve avere estensione .u. Ad esempio per caricare il file demo.u, ci sono due modi alternativi: o premendo su Select a file e dopo aver scelto il file, si preme su Send oppure inviando il comando:

```
load("demo.u");
```

5.2 Interfacciamento dei sensori non supportati dal linguaggio URBI

5.2.1 Sensori non supportati inizialmente

Nella piattaforma URBI sono nativamente interfacciati i sensori del kit Lego Mindstorms NXT mentre non sono disponibili gli oggetti per l'uso dei sensori opzionali. La stragrande maggioranza dei sensori non supportati da URBI sono sensori digitali. Più precisamente, dei sensori che il dipartimento possiede, solamente il giroscopio non è un sensore digitale. I sesnori digitali utilizzano il bus I^2C.

5.2.2 Il bus I^2C e il suo protocollo di comunicazione

I^2C è un bus di controllo che fornisce il collegamento tra circuiti integrati in un sistema. Questo bus è di tipo seriale e, attraverso due linee dati, SDA (Serial DAta line) e SCL (Serial Clock Line), trasmette le informazioni a tutti i dispositivi ad esso collegati. Ogni dispositivo è identificato da un indirizzo univoco e può operare o come ricevitore o come trasmettitore in base alle funzioni che esso svolge. Ciascun dispositivo deve inoltre essere impostato o a master o a slave: il dispositivo che inizia la comunicazione generando il segnale di clock per la sincronizzazione deve essere impostato a master mentre tutti gli altri sono impostati a slave. E' possibile che nel bus vi siano più dispositivi master, ma in tal caso solamente uno alla volta può controllare il bus. Le specifiche del bus I^2C prevedono tre tipi di comunicazione: singolo messaggio da master a slave, singolo messaggio da slave a master o combinazione di messaggi.

Il brick NXT è dotato di quattro bus I^2C, uno per ogni porta di input. Nella comunicazione digitale il brick NXT può assumere solamente la funzione di master; proprio per questo non si possono far comunicare due brick per mezzo del bus I^2C. La velocità con cui il brick trasferisce i dati sul bus è di

9600 bit/secondo. Ogni canale dispone di un buffer da 16 byte, perciò il brick NXT può spedire e ricevere al massimo 16 byte per ciclo di trasmissione.

5.2.3 Il protocollo dei comandi diretti del robot didattico Lego Mindstorms NXT

Il protocollo dei comandi diretti del Lego Mindstorms NXT, inviati via Bluetooth o via USB, rende possibile il controllo del robot da dispositivi esterni che possono essere un altro brick NXT, un personal computer oppure un qualsiasi dispositivo bluetooth che usi il profilo seriale per le porte. Lo scopo di questi comandi diretti è quello di fornire un'interfaccia di controllo per utilizzare le funzionalità del brick da un dispositivo esterno senza che sia necessario scrivere e/o eseguire un programma di controllo remoto specializzato nel NXT. La struttura dei comandi diretti e delle relative risposte è la seguente:

Byte 0	Byte 1	Byte 2	Byte 3				Byte N

nella quale il byte 0 specifica il tipo di comando che può essere:

- 0x00: comando diretto con richiesta di risposta
- 0x01: comando di sistema con richiesta di risposta
- 0x02: risposta
- 0x80: comando diretto senza richiesta di risposta
- 0x81: comando di sistema senza richiesta di risposta

Dal byte 1 al byte N sono contenute le informazioni riguardanti o il comando inviato o la risposta ricevuta. La dimensione massima dei comandi diretti è fissata a 64 byte includendo il byte 0. Se il comando è con richesta di risposta, la latenza stimata è approssimativamente di 60 ms. La documentazione dei comandi diretti è disponibile nel pacchetto Lego Mindstorms NXT Bluetooth Developer Kit. Si riportano le tabelle che descrivono i comandi diretti e i loro parametri.

Il comando STARTPROGRAM fa eseguire sul NXT il programma il cui nome è specificato come parametro e riceve come risposta dal NXT un byte

Tabella 5.1: Struttura del comando STARTPROGRAM

Byte 0	0x00 o 0x80	
Byte 1	0x00	
Byte 2-21	File name	Nome del file da eseguire in formato ASCIIZ

Tabella 5.2: Struttura della risposta al comando STARTPROGRAM

Byte 0	0x02	
Byte 1	0x00	
Byte 2	Status Byte	se >0 segnala che ci sono stati degli errori

che indica se ci sono stati errori.

Il comando STOPPROGRAM fa arrestare il programma in esecuzione sul NXT e riceve come risposta un byte che indica se ci sono stati errori.

Tabella 5.3: Struttura del comando STOPPROGRAM

Byte 0	0x00 o 0x80
Byte 1	0x01

Tabella 5.4: Struttura della risposta al comando STOPPROGRAM

Byte 0	0x02	
Byte 1	0x01	
Byte 2	Status Byte	se >0 segnala che ci sono stati degli errori

Il comando PLAYSOUNDFILE fa riprodurre un file musicale al NXT e riceve come risposta un byte che indica se ci sono stati errori.

Tabella 5.5: Struttura del comando PLAYSOUNDFILE

Byte 0	0x00 o 0x80	
Byte 1	0x02	
Byte 2	Loop?	valore booleano che indica se il file musicale deve essere ripetuto all'infinito o meno
Byte 3-22	File name	Nome del file da eseguire in formato ASCIIZ

Tabella 5.6: Struttura della risposta al comando PLAYSOUNDFILE

Byte 0	0x02	
Byte 1	0x02	
Byte 2	Status Byte	se >0 segnale che ci sono stai degli errori

Il comando PLAYTONE fa riprodurre un suono ad una frequenza passata come parametro e per un intervallo di tempo passato come parametro. Riceve come risposta un byte che indica se ci sono stati errori.

Tabella 5.7: Struttura del comando PLAYTONE

Byte 0	0x00 o 0x80	
Byte 1	0x03	
Byte 2	Frequency	frequenza del suono da ripordurre nel range che va da 200 a 14000
Byte 3-22	File name	Nome del file da eseguire in formato ASCIIZ

Tabella 5.8: Struttura della risposta al comando PLAYTONE

Byte 0	0x02	
Byte 1	0x03	
Byte 2	Status Byte	se >0 segnale che ci sono stai degli errori

Il comando SETOUTPUTSTATE imposta tutti i parametri di una porta di output (per esempio per comandare un servomotore) e riceve come risposta un byte che indica se ci sono stati errori.

Il comando SETINPUTMODE imposta i parametri di funzionamento del sensore collegato alla porta selezionata

Il comando GETOUTPUTSTATE richiede i parametri con la quale è stata settata una certa porta e riceve tutte le informazioni da quella porta.

Tabella 5.9: Struttura del comando SETOUTPUTSTATE

Byte 0	0x00 o 0x80	
Byte 1	0x04	
Byte 2	Output port	porta da settare (le porte sono numerate da 0 a 2 mentre 255 indica di settare tutte le porte allo stesso modo)
Byte 3	Power set point	potenza di uscita della porta nel range che va da -100 a 100
Byte 4	Mode byte	modo con cui impostare la porta. I modi disponibili sono descritti nella tabella 5.11
Byte 5	Regulation mode	modo di regolazione della porta. I modi di regolazione che si possono impostare sono descritti nella tabella 5.12
Byte 6	Turn ratio	valore da -100 a 100 : imposta il verso di rotazione dei motori
Byte 7	Run state	Stato in cui settare il motore. Gli stati disponibili sono descritti nella tabella 5.13
Byte 8-12	TachoLimit	numero massimo di giri che il motore può eseguire: se impostato a 0 gira all'infinito

Tabella 5.10: Struttura della risposta al comando SETOUTPUTSTATE

Byte 0	0x02	
Byte 1	0x04	
Byte 2	Status Byte	se >0 segnale che ci sono stai degli errori

Tabella 5.11: MODE - Modalità delle porte di output

MOTORON	0x01	attiva il motore specificato
BRAKE	0x02	ferma il motore specificato
REGULATED	0x03	attiva la modalità regolata per il motore specificato

Tabella 5.12: REGULATION MODE - Modalità di regolazione

REGULATION_MODE_IDLE	0x00	nessuna regolazione abilitata
REGULATION_MODE_MOTOR_SPEED	0x01	la regolazione di potenza viene abilitata per il motore selezionato
REGULATION_MODE_MOTOR_SYNC	0x02	la sincronizzazione viene abilitata (questa modalità deve essere settata in almeno due porte perchè funzioni)

Tabella 5.13: RUN STATE - Stati dei motori

MOTOR_RUN_STATE_IDLE	0x00	il motore viene disabilitato
MOTOR_RUN_STATE_RAMPUP	0x10	il motore viene attivato in modo crescente
MOTOR_RUN_STATE_RUNNING	0x20	il motore è in modalità running
MOTOR_RUN_STATE_RAMPDOWN	0x40	il motore viene disabilitato in modo decrescente

Tabella 5.14: Struttura del comando SETINPUTMODE

Byte 0	0x00 o 0x80	
Byte 1	0x05	
Byte 2	Input Port	specifica su quale porta è connesso il sensore e può assumere un valore compreso tra 0 e 3
Byte 3	Sensor Type	specifica il tipo di sensore. I tipi di sensore disponibili sono descritti nella tabella 5.16
Byte 4	Sensor Mode	specifica il modo di funzionamento del sensore. I modi di funzionamento sono descritti nella tabella 5.17

Tabella 5.15: Struttura della risposta al comando SETINPUTMODE

Byte 0	0x02	
Byte 1	0x05	
Byte 2	StatusByte	se >0 segnala che ci sono stati degli errori

Tabella 5.16: SENSOR TYPE - Tipi di Sensori

NO_SENSOR	0x00
SWITCH	0x01
TEMPERATURE	0x02
REFLECTION	0x03
ANGLE	0x04
LIGHT_ACTIVE	0x05
LIGTH_INACTIVE	0x06
SOUND_DB	0x07
SOUNDDBA	0x08
CUSTOM	0x09
LOWSPEED	0x0A
LOWSPEED_9V	0x0B
NO_OF_SENSOR_TYPE	0x0C

Tabella 5.17: SENSOR MODE

RAWMODE	0x00
BOOLEANMODE	0x20
TRANSITIONCNTMODE	0x40
PERIODCOUNTERMODE	0x60
PCTFULLSCALEMODE	0x80
CELSIUSMODE	0xA0
FAHRENHEITMODE	0xC0
ANGLESTEPSMODE	0xE0
SLOPEMASK	0x1F
MODEMASK	0xE0

Tabella 5.18: Struttura del comando GETOUTPUTSTATE

Byte 0	0x00 o 0x80
Byte 1	0x06
Byte 2	Output Port

Tabella 5.19: Struttura della risposta al comando GETOUTPUTSTATE

Byte 0	0x02	
Byte 1	0x06	
Byte 2	StatusByte	se >0 segnale che ci sono stati errori
Byte 3	Output Port	segnala su quale porta sono stati letti i dati
Byte 4	Power set point	Potenza settata sulla porta: è un valore compreso tra -100 e 100
Byte 5	Mode	Modo in cui è stata settata la porta
Byte 6	Regulation Mode	Modo di regolazione selezionato per questa porta
Byte 7	Turn ratio	modalità di rotazione impostata
Byte 8	Run State	Stato in cui si trova la porta selezionata
Byte 9-12	TachoLimit	numero massimo di giri impostati che il motore dovrà fare
Byte 13-16	TachoCount	numero di giri effettuati dal motore dopo l'ultimo reset del contatore del motore
Byte 17-20	BlockTachoCount	posizione riferita in numero di giri relativi all'ultimo movimento del motore
Byte 21-24	RotationCount	posizione riferita in numero di giri relativi dall'ultimo reset del sensore di rotazione del motore

Il comando GETINPUTVALUES richiede la lettura dei dati da una porta di input. Questo comando riceve come risposta i dati oppure una segnalazione d'errore.

Tabella 5.20: Struttura del comando GETINPUTVALUES

Byte 0	0x00 o 0x80
Byte 1	0x07
Byte 2	Input Port

Il comando RESETINPUTSCALEDVALUE effettua il reset di una porta di input e riceve come risposta un byte che segnala se ci sono stati degli errori.

Tabella 5.21: Struttura della risposta al comando GETINPUTVALUES

Byte 0	0x02	
Byte 1	0x07	
Byte 2	StatusByte	se >0 segnale che ci sono stati errori
Byte 3	Input Port	segnala su quale porta sono stati letti i dati
Byte 4	Valid?	seganla se i dati ottenuti sono validi o meno
Byte 5	Calibrated?	segnala se è stato trovato un file di calibrazione del sensore
Byte 6	Sensor Type	segnale quale tipo di sensore era stato settato
Byte 7	Sensor Mode	segnale il modo di funzionamento del sensore settato precedentemente
Byte 8-9	Raw A/D value	valore raw
Byte 10-11	Normalized A/D value	valore normalizzato in modo che sia compreso tra 0 e 1023
Byte 12-13	Scaled value	valore scalato in base al modo settato
Byte 14-15	Calibrated value	valor scalato in base alla calibrazione. Solitamente non utilizzato

Tabella 5.22: Struttura del comando RESETINPUTSCALEDVALUE

Byte 0	0x00 o 0x80
Byte 1	0x08
Byte 2	Input Port

Tabella 5.23: Struttura della risposta al comando RESETINPUTSCALED-VALUE

Byte 0	0x02	
Byte 1	0x08	
Byte 2	StatusByte	se >0 segnale che ci sono stati errori

Il comando RESETMOTORPOSITION effetua il reset della porta di output e riceve come risposta un byte che indica se ci sono stati errori.

Il comando GETBATTERYLEVEL richiede al brcik le informazioni riguardanti lo stato della batteri e riceve come risposta il valore della tensione in

Tabella 5.24: Struttura del comando RESETMOTORPOSITION

Byte 0	0x00 o 0x80	
Byte 1	0x0A	
Byte 2	Output Port	
Byte 3	Relative?	valore booleano che indica se la posizione da resettare è quella relativa all'ultimo movimento o meno.

Tabella 5.25: Struttura della risposta al comando RESETMOTORPOSITION

Byte 0	0x02	
Byte 1	0x0A	
Byte 2	StatusByte	se >0 segnale che ci sono stati errori

millivolt oppure una segnalazione d'errore.

Tabella 5.26: Struttura del comando GETBATTERYLEVEL

Byte 0	0x00 o 0x80
Byte 1	0x0B

Tabella 5.27: Struttura della risposta al comando GETBATTERYLEVEL

Byte 0	0x02	
Byte 1	0x0B	
Byte 2	StatusByte	se >0 segnale che ci sono stati errori
Byte 3	Voltage in millivolts	indica la tensione in millivolt

Il comando STOPSOUNDPLAYBACK arresta l'esecuzione di un file sonoro e riceve come risposta lo stato che indica se ci sono stati errori.

Tabella 5.28: Struttura del comando STOPSOUNDPLAYBACK

Byte 0	0x00 o 0x80
Byte 1	0x0C

Il comando KEEPALIVE manda in modalità sleep il brick NXT

Tabella 5.29: Struttura della risposta al comando STOPSOUNDPLAY-BACK

Byte 0	0x02	
Byte 1	0x0C	
Byte 2	StatusByte	se >0 segnale che ci sono stati errori

Tabella 5.30: Struttura del comando STOPSOUNDPLAYBACK

Byte 0	0x00 o 0x80
Byte 1	0x0D

Tabella 5.31: Struttura della risposta al comando STOPSOUNDPLAY-BACK

Byte 0	0x02	
Byte 1	0x0D	
Byte 2	StatusByte	se >0 segnale che ci sono stati errori
Byte 3-6	Sleep time limit	Tempo massimo in millisecondi in cui il robot rimane in modalità sleep

Il comando LSGETSTATUS richiede lo stato dei dati del I^2C e riceve come risposta il numero di byte disponibili e un byte che segnala se ci sono stati degli errori.

Tabella 5.32: Struttura del comando LSGETSTATUS

Byte 0	0x00 o 0x80
Byte 1	0x0E
Byte 2	Port

Tabella 5.33: Struttura della risposta al comando LSGETSTATUS

Byte 0	0x02	
Byte 1	0x0E	
Byte 2	StatusByte	se >0 segnale che ci sono stati errori
Byte 3	Bytes Ready	segnala quanti byte sono pronti ad essere letti

Il comando LSWRITE richiede al brick dati provenienti dalle porte settate in Low Speed I^2C.

Tabella 5.34: Struttura del comando LSWRITE

Byte 0	0x00 o 0x80	
Byte 1	0x0F	
Byte 2	Port	
Byte 3	TX Data Length	Lunghezza dei dati da trasmettere in byte
Byte 4	RX Data Length	Lunghezza dei dati da ricevere in byte
Byte 5-N	Data	Dati da trasmettere, dove N= TX Data Length + 4

Tabella 5.35: Struttura della risposta al comando LSWRITE

Byte 0	0x02	
Byte 1	0x0F	
Byte 2	StatusByte	se >0 segnale che ci sono stati errori

Il comando LSREAD richiede la lettura dei dati da una porta di input I^2C e riceve come risposta i dati provenienti dal bus.

Tabella 5.36: Struttura del comando LSREAD

Byte 0	0x00 o 0x80
Byte 1	0x10
Byte 2	Port

Tabella 5.37: Struttura della risposta al comando LSREAD

Byte 0	0x02	
Byte 1	0x10	
Byte 2	StatusByte	se >0 segnale che ci sono stati errori
Byte 3	Bytes Read	numero di byte letti
Byte 4-19	RX Data	dati ricevuti

Il comando GETCURRENTPROGRAMNAME richiede il nome del programma attualmente in esecuzione nel brick e riceve come risposta proprio tale nome.

Tabella 5.38: Struttura del comando GETCURRENTPROGRAMNAME

Byte 0	0x00 o 0x80
Byte 1	0x11

Tabella 5.39: Struttura della risposta al comando GETCURRENTPROGRAMNAME

Byte 0	0x02	
Byte 1	0x11	
Byte 2	StatusByte	se >0 segnale che ci sono stati errori
Byte 3-22	File name	nome del programma attualmente in esecuzione

In tutte le risposte ai comandi illustrati precedentemente si trova lo StatusByte. Se questo byte è uguale a zero nonc i sono stati errori altrimenti riportiamo la lista degli errori che si possono incontrare durante l'utilizzo dei comandi diretti.

5.2.4 Codice URBI per l'interfacciamento dei sensori non supportati inizialmente

Per interfacciare i sensori non supportati dalla piattaforma, si è creato un file sensor.u, da caricare direttamente dalla console, nel quale sono contenute le funzioni per impostare i parametri di funzionamento delle porte di input, per richiedere dati da un sensore e per leggere i dati dal bus. Poiché il linguaggio urbiScript, incluso nella distribuzione URBI for Lego Mindstorms NXT 1.5, non supporta i numeri esadecimali, si è innanzi tutto reso indispensabile convertire gli opCode da esadecimale a decimale. La piattaforma URBI-sdk-2.0, che è appena stata rilasciata, invece supporta i numeri esadecimali ma

Tabella 5.40: Valori dello StatusByte e corrispettivi errori

0x20	In attesa di una comunicazione in corso
0xBD	La richiesta non può essere soddifatta (per esempio il file non è stato trovato)
0xBE	Opcode del comando sconosciuto
0xBF	Pacchetto danneggiato
0xC0	I dati contengono valori fuori dal range
0xDD	Errore sul bus di comunicazione
0xDE	Memoria esaurita nel buffer della comunicazione
0xDF	La connessione specificata non è valida
0xE0	La connessione specificata non è configurata o è occupata
0xEC	Nessun programma attivo
0xED	Dimensione specificata non corretta
0xEF	Tentativo di accesso ad un campo o una struttura invalidi
0xF0	Inputo o Output specificati invalidi
0xFB	Memoria disponibile insufficiente
0xFF	Argomento errato

non è al momento interfacciabile con il robot NXT. Inoltre la piattaforma è munita di due metodi per inviare i comandi diretti e i comandi diretti con risposta richiesta, questi sono corrispondentemente:

```
Command.send(bufferOpCode);
```

e

```
var answer=Command.request(bufferOpCode,size);
```

Il parametro bufferOpCode è un array contenente gli opCode convertiti in numeri interi compresi tra 0 e 255. Il parametro size invece è la dimensione del buffer della risposta che si deve ottenere. La variabile answer sarà invece un array contenente la risposta al comando. In merito alla dimensione dei buffer, c'e una cosa importante da notare: il primo byte del comando, che indica se si tratta di un comando diretto con o senza risposta richiesta viene omesso nel bufferOpCode ed anche il primo byte della risposta che indica che il buffer è un buffer di risposta, non viene riportato. Di conseguenza quando si usano i comandi diretti nella piattaforma URBI for Lego Mindstorms NXT è neccessario rimuovere il primo byte e tener conto di questo per le dimensioni dei buffer, dato che se si sbaglia ad impostare il parametro size nel comando con richiesta di risposta, si riceve un buffer vuoto.
Ad esempio, per far riprodurre un suono all'altoparlante del robot senza usare l'oggetto Beeper si può inviare il seguente comando:

```
Command.send([3,10,10,0,0]);
```

Ancora, per richiedere le informazioni riguardanti la porta 1 si deve digitare il comando :

```
var answer=Command.request([7,0],15);
```

Il codice seguente riporta il listato del file sensor.u :

```
//PORTE IN INGRESSO
var PORT1 = 0 ;
var PORT2 = 1 ;
var PORT3 = 2 ;
var PORT4 = 3 ;

//SENSORTYPE
var NOSENSOR = 0 ;
var CUSTOM = 9 ;
var LOWSPEED = 10 ;
var LOWSPEED9V = 11 ;

//SENSORMODE
var RAW = 0 ;
var BOOLEANMODE = 32 ;

//US_SENSOR_MODE
var US_ADDRESS = 2 ;
var SET_US_MODE = 65 ;
var READ_US_BYTE0 = 66 ;
var READ_US_BYTE1 = 67 ;
var READ_US_BYTE2 = 68 ;
var READ_US_BYTE3 = 69 ;
var READ_US_BYTE4 = 70 ;
var READ_US_BYTE5 = 71 ;
var SET_US_CONTINUOUSINTERVAL = 64 ;
var US_MODE_OFF = 0 ;
var US_MODE_SINGLESHOT = 1 ;
var US_MODE_CONTINUOUS = 2 ;
var US_MODE_EVENTCAPTURE = 3 ;

//COMANDI
var SETOUTPUTSTATE = 4 ;
var SET_INPUT_MODE = 5 ;
var GETOUTPUTSTATE = 6 ;
var GETINPUTVALUES = 7 ;
var RESETMOTORPOSITION = 10 ;
```

```
var LSGETSTATUS = 14 ;
var LSWRITE = 15 ;
var LSREAD = 16 ;


//FUNZIONE PER IMPOSTARE IL TIPO DI SENSORE COLLEGATO AD
   UNA PORTA
function setInputMode( porta , sensorType , sensorMode ){
        var a = Command.request ( [ SET_INPUT_MODE , porta
           , sensorType , sensorMode ] , 2 ) ;
        a ;
        if ( a[1]!=0 ){
           echo ( "Errore: setInputMode non riuscita" ) ;
        }else{
           echo ( "SetInputMode riuscita" ) ;
        };
        return a;
};

//FUNZIONE PER SELEZIONARE LA MODALITA' DI ACQUISIZIONE DEI
    DATI DAL SENSORE
//porta: seleziona la porta alla quale il sensore è conneso
//usMode: modo in cui settare il sensore : US_MODE_OFF ,
   US_MODE_SINGLESHOT , US_MODE_CONTINUOUS ,
   US_MODE_EVENTCAPTURE
function setUSMode ( porta , usMode ){
         var e = Command.request ( [ LSWRITE , porta , 3 ,
            0 , US_ADDRESS , SET_US_MODE , usMode  ] , 2) ;
         e;
         if ( e[1]!=0 ){
            echo ( "Errore: setUSMode non riuscita" ) ;
         }else{
            echo ( "SetUSMode riuscita" ) ;
         };
         return e;
};

//FUNZIONE PER LA LETTURA DI UN SENSORE DIGITALE
//porta: seleziona la porta alla quale il sensore è
   connesso
function getI2CSensorVal ( porta ) {
         var b = Command.request ( [ LSWRITE , porta , 2 ,
            1 , US_ADDRESS , READ_US_BYTE0  ] , 2 ) ;
```

```
        echo (b);

        var i=0;
        var byteReady = 0 ;
        var status = 0 ;
        while ( i < 20 && byteReady < 1 ) {
              var h = Command.request ( [ LSGETSTATUS ,
                  porta ] , 3 );
              h ;
              i++ ;
              if ( h != [] ){
                  status = h[1] ;
                  //echo ( "Status: " + status ) ;
                  if ( status == 0 )
                      byteReady = h[2] ;
                      //echo ( "ByteReady: "+byteReady ) ;
              };
              wait (500ms);
        };
        if ( status != 0 ){
           echo ( "Errore sul bus I2C" ) ;
        }else{
           //echo ( "ByteReady: " + byteReady );
           var c = Command.request ( [ LSREAD , porta ] ,
              19 ) ;
           var statusb = c[1];
           if ( statusb != 0 ){
              echo ( "Errore in lettura sul bus I2C" );
           }else{
              echo (c);
              return c;
           };
        };
};

//FUNZIONE PER LA LETTURA DI UN SENSORE ACCELLEROMETRO
//porta: seleziona la porta alla quale il sensore è
   connesso
function getAccelerometerVal ( porta ) {
        var a = Command.request ( [ LSWRITE , porta , 2 ,
           1 , US_ADDRESS , READ_US_BYTE0  ] , 2 ) &
        var b = Command.request ( [ LSWRITE , porta , 2 ,
           1 , US_ADDRESS , READ_US_BYTE1  ] , 2 ) &
```

```
        var c = Command.request ( [ LSWRITE , porta , 2 ,
            1 , US_ADDRESS , READ_US_BYTE2 ] , 2 ) &
        var d = Command.request ( [ LSWRITE , porta , 2 ,
            1 , US_ADDRESS , READ_US_BYTE3 ] , 2 ) &
        var e = Command.request ( [ LSWRITE , porta , 2 ,
            1 , US_ADDRESS , READ_US_BYTE4 ] , 2 ) &
        var f = Command.request ( [ LSWRITE , porta , 2 ,
            1 , US_ADDRESS , READ_US_BYTE5 ] , 2 ) ;

        var i=0;
        var byteReady = 0 ;
        var status = 0 ;
        while ( i < 20 && byteReady < 1 ) {
                var h = Command.request ( [ LSGETSTATUS ,
                    porta ] , 3 );
                h ;
                i++ ;
                if ( h != [] ){
                    status = h[1] ;
                    //echo ( "Status: " + status ) ;
                    if ( status == 0 )
                        byteReady = h[2] ;
                        //echo ( "ByteReady: "+byteReady ) ;
                };
                wait (500ms);
        };
        if ( status != 0 ){
            echo ( "Errore sul bus I2C" ) ;
        }else{
            //echo ( "ByteReady: " + byteReady );
            var c = Command.request ( [ LSREAD , porta ] ,
                19 ) ;
            var statusb = c[1];
            if ( statusb != 0 ){
                echo ( "Errore in lettura sul bus I2C" );
            }else{
                echo (c);
                return c;
            };
        };
};

//FUNZIONE PER LA LETTURA DI UN SENSORE ANALOGICO
```

```
//porta: seleziona la porta sulla quale il sensore è
    connesso
function getAnalogicSensorVal ( porta ){
        var k = Command.request ([ GETINPUTVALUES , porta
            ],15) ;
        return k;
} ;
```

Per testere il funzionamento dei senori analogici si è provato il giroscopio (Gyro Sensor) e si è creato il file gyro.u contente il codice per provarlo.

```
//uso la porta 3 del NXT
//IMPOSTO LA PORTA IN MODALITA' LOWSPEED E I DATI IN
    MODALITA' RAW
setInputMode( PORT3 , NO_SENSOR , RAW );
//LEGGO I DATI DAL SENSORE
var v = getAnalogicSensorVal ( PORT3 )
//RICAVO LA MISURAZIONE DAL BUFFER
var byte7 = v[7];
var byte8 =v[8];
var mis=(byte7*4)+byte8;
echo ("Il valore letto dal sensore è: "+mis);
```

Per testare il funzionamento dei sensori digitali è stato utilizzato il sensore ad ultrasuoni, che come già detto è un sensore I^2C. Il codice seguente utilizza le funzioni sviluppate precedentemente per leggere il valore della distanza percepita dal sensore.

```
//uso la porta 4 del NXT
//IMPOSTO LA PORTA IN MODALITA' LOWSPEED E I DATI IN
    MODALITA' RAW
setInputMode( PORT4 , LOWSPEED9V , RAW );
//IMPOSTO LA MODALITA' DI RILEVAZIONE DEI DATI DEL SENSORE
    IN CONTINUA
setUSMode ( PORT4 , US_MODE_CONTINUOUS );
//LEGGO I DATI DAL SENSORE
var v = getI2CSensorVal ( PORT4 );
//RICAVO LA MISURAZIONE DAL BUFFER
var mis = v[3];
echo ("Il valore letto dal sensore è: "+mis);
```

Le stesse operazioni possono essere eseguite per leggere il colorNumber rilevato dal sensore di colori (Color Sensor) e i gradi misurati dalla bussola (Compass Sensor).
Per leggere i valori rilevati dall'accelerometro (Accelerometer/Tilt Sensor) invece si è creato il file acellerometer.u della quale si riporta il contenuto:

```
//uso la porta 4 del NXT
//IMPOSTO LA PORTA IN MODALITA' LOWSPEED E I DATI IN
    MODALITA' RAW
setInputMode( PORT4 , LOWSPEED9V , RAW );
//IMPOSTO LA MODALITA' DI RILEVAZIONE DEI DATI DEL SENSORE
    IN CONTINUA
setUSMode ( PORT4 , US_MODE_CONTINUOUS );
//LEGGO I DATI DAL SENSORE
var v = getAccelerometerVal ( PORT4 );
//RICAVO LA MISURAZIONE DAL BUFFER
var x = v[3];
var y = v[4];
var z = v[5];
if (x>127) x-=256
x=x*4+v[6]
if (y>127) y-=256
y=y*4+v[7]
if (z>127) z-=256
z=z*4+v[8]
echo ("Il valore dell'asse x è: "+x);
echo ("Il valore dell'asse y è: "+y);
echo ("Il valore dell'asse z è: "+z);
```

I valori ottenuti con questo codice sono compresi tra 0 e 255, quindi devono essere scalati opportunamente.

Capitolo 6

Dimostrazione di stabilità del sistema URBI - NXT

6.1 Disabilitazione del PID dei servomotori

I servomotori del robot NXT sono controllati da un PID. Per disabilitare questo PID è neccessario pilotarli con i comandi diretti in modo da impostare i parametri manualmente. Per far in modo che il motore si muova senza far intervenire nei suoi movimenti il controllore ad esso associtato ho usato il comando SETOUTPUTSTATE, impostando il parametro Regulation Mode in REGULATION_MODE_IDLE. Più in particolare ho dovuto inviare il seguente comando:

```
var x = Command.request ( 0 , SETOUTPUTSTATE , PORTA ,
    POWER , MOTORON , REGUALTION_MODE_IDLE , 0 ,
    MOTOR_RUN_STATE_RUNNING , 0 ) ;
```

dove SETOUTPUTSTATE=4, PORTA specifica la porta alla quale è collegato il motore, POWER specifica la potenza del motore in un range (-100 , 100), MOTORON=1,
REGULATION_MODE_IDLE=0,
MOTOR_RUN_STATE_RUNNING=32.

In questo modo i motori si muovono solamente in base ai comandi che vengono loro inviati e non vengono retroazionati dal PID.
Per controllare i motori ho crato il file motor.u del quale si riporta il contenuto:

```
//PORTE IN USCITA
var OUT1 = 0 ;
var OUT2 = 1 ;
var OUT3 = 2;

//COMANDI
var SETOUTPUTSTATE = 4 ;
var SET_INPUT_MODE = 5 ;
var GETOUTPUTSTATE = 6 ;
var GETINPUTVALUES = 7 ;
var RESETMOTORPOSITION = 10 ;
var LSGETSTATUS = 14 ;
var LSWRITE = 15 ;
var LSREAD = 16 ;

//MODALITA' MOTORI
var MOTORON = 1 ;
var BRAKE = 2 ;
var REGULATED =4 ;

//MODI DI REGOLAZIONE MOTORI
var REGULATION_MODE_IDLE = 0 ;
var REGULATION_MODE_MOTOR_SPEED = 1 ;
var REGULATION_MODE_MOTOR_SYNC = 2;

//MODI IN RUNSTATE MOTORI
var MOTOR_RUN_STATE_IDLE = 0 ;
var MOTOR_RUN_STATE_RAMPUP = 16 ;
var MOTOR_RUN_STATE_RUNNING = 32 ;
var MOTOR_RUN_STATE_RAMPDOWN = 64;

//FUNZIONE CHE ATTIVA IL MOTORE CON UNA CERTA POTENZA
//porta: porta sulla quale è collegato il motore
//power: valore compreso tra 0 e 255 che indica la potenza
    del motore: tra 0 e 127 avanti (127 = 100% power) e tra
    128 e 255 indietro ( 128=100% power e 255=0% power)
//tachoCount: numero assoluto di gradi al motore e deve
    essere >= 0
//mode : indica il modo di funzionamento del motore e può
    essere: MOTORON, BRAKE, REGULATED
//regulationMode : indica la modalità di regolazione del
    motore e può essere: REGULATION_MODE_IDLE ,
    REGULATION_MODE_MOTOR_SPEED , REGULATION_MODE_MOTOR_SYNC
```

```
//motorState : Indica lo stato del motore e può essere :
   MOTOR_RUN_STATE_IDLE , MOTOR_RUN_STATE_RAMPUP ,
   MOTOR_RUN_STATE_RAMPDOWN , MOTOR_RUN_STATE_RUNNING
function setMotorOn ( porta , power , tachoCount , mode ,
   regulationMode , motorState ){
        echo ("POWER: " + power);
        if ( power < 0 || power > 255 ){
           echo ( "Il valore immesso della potenza non è
              corretto" ) ;
           return ;
        };
        if ( tachoCount < 0 ){
           echo ( "Il valore immesso dei gradi di
              rotazione non è corretto " );
           return;
        };
        var  byte8 = 0;
        var  byte9 = 0;
        if ( tachoCount > 256 ){
           byte9 = tachoCount / 256 ;
           tachoCount = tachoCount - ( byte9 * 256 ) ;
        };
        if (tachoCount > 0){
           byte8 = tachoCount ;
        };
        var h = Command.request ( [ SETOUTPUTSTATE , porta
            , power , mode , regulationMode , 0 ,
           motorState , byte8 , byte9 , 0 , 0 , 0 ] , 2 );
        h;
};

//FUNZIONE CHE ARRESTA IL MOTORE
//porta: porta sulla quale è collegato il motore
//brake: segnala se attivare o meno i freni e puo essere
   false (freni disattivati) o true (freni attivati)
//mode : indica il modo di funzionamento del motore e può
   essere: MOTORON, BRAKE, REGULATED
function stopMotor( porta , brake , mode){
         if ( brake ) {
            var y = Command.request ( [ SETOUTPUTSTATE ,
               porta , 0 , mode ,
               REGULATION_MODE_MOTOR_SPEED , 0 ,
               MOTOR_RUN_STATE_RUNNING , 0 , 0 , 0 , 0 ,0
```

```
                ] , 2 ) ;
            y;
        }else{
            var y = Command.request ( [ SETOUTPUTSTATE ,
                porta , 0 , 0 , REGULATION_MODE_IDLE , 0 ,
                MOTOR_RUN_STATE_IDLE , 0 , 0 , 0 , 0 , 0 ]
                , 2 );
            y;
        };
};

//FUNZIONE CHE RESETTA LA POSIZIONE IN GRADI DEL MOTORE
//porta: porta sulla quale è collegato il motore
//relative: valore booleano che se false resetta il numero
    di giri dall'ultimo reset della posizione e
//se true resetta il numero di giri effettusti dopo l'
    ultimo movimento
function resetMotorPosition ( porta , relative ){
        var y = Command.request ( [ RESETMOTORPOSITION ,
            porta , relative ] , 2 );
        echo y;
};

//FUNZIONE CHE RESTITUISCE LA POSIZIONE DEL MOTORE IN
    ROTAZIONI
//porta : porta sulla quale è collegato il motore
//relative : valore booleano che indica se si vuole
    ottenere il numero di rotazioni nell'ultimo movimento (
    true) o dall'ultimo reset (false)
function getMotorRotation ( porta , relative ){
        var rot;
        var k = Command.request ([6 , porta ] , 24 );
        if ( relative ) {
            rot = k[19]*16777216 + k[18]*65536 + k[17]*256
                + k[16] ;
        }else{
            rot = k[23]*16777216 + k[22]*65536 + k[21]*256
                + k[20] ;
        };
        return rot ;
};
```

La funzione setMotorOn permette il controllo dei servomotori impostando tutti i parametri di funzionamento; in particolare, per ottenere il comando

precedente, si deve digitare:

```
setMotorOn ( PORTA , POWER , 0 , MOTORON ,
    REGUALTION_MODE_IDLE , MOTOR_RUN_STATE_RUNNING )
```

La funzione stopMotor invece arresta il movimento dei servomotori specificando se devono essere usati i freni e la modalità in cui il motore si trova dopo l'arresto.
Invece la funzione resetMotorPosition l'ho creata per reimpostareil contatore dei giri del motore, specificando se devono essere azzerati tutti i giri effetuati dal motore o solamente quelli relativi all'utlimo movimento del motore.
Infine la funzione getMotorRotation restituisce il valore del contatore dei giri del motore relativi o all'ultimo movimento o relativi a tutti i movimenti effettuati dopo l'ultimo reset.

6.2 Esperimento

Ho deciso di preparare un esperimento per valutare la stabilità del sistema URBI-NXT poiché l'utilizzo della piattaforma URBI per controllare il robot Lego NXT comporta dei ritardi. Questi ritardi rendono il sistema instabile se le richieste hanno dei tempi troppo stringenti rispetto ad essi. Il robot utilizzato in questa prova è montato secondo il modello tribot senza pinze anteriori, con il claw (ruota sterzante posteriore)bloccato; il robot è munito solamente del sensore ultrasuoni. L'esperimento consiste nel posizionare il robot di fronte ad una parete e, fissata come target una certa distanza, ho fatto muovere il robot variando la potenza dei motori (il robot si deve muovere lungo una linea retta perpendicolare alla parete) ed ho registrato se il robot raggiunge il target fissato fermandosi su di esso. In particolare, ho fatto muovere il robot in avanti, con la potenza fissata, se la distanza rilevata dal sensore ultrasuoni è superiore al target o ho fatto muovere il robot all'indietro se la distanza misurata è inferiore al target.

Il risultato di questo esperimento è però influenzato da molti fattori tra i quali la carica della batteria è uno dei principali. Infatti la potenza richiesta ai motori è espressa in percentuale della potenza massima, ma la potenza massima dipende dalla tensione applicata ai servomotori e quindi dalla carica della batteria. Un altro fattore che influenza molto l'esperimento è la superficie del piano su cui si fa muovere il robot: se lo si appoggia sul pavimento, le fessure delle piastrelle creano dei movimenti non previsti deviando, anche se di poco, la traiettoria del robot mentre se lo si appoggia su un piano liscio, come ad esempio un tavolo, il robot mantiene molto più facilmente la traiettoria scelta. Infine anche la distanza fissata come target comporta

delle variazioni dei risultati: il sensore ultrasuoni ha una tolleranza di circa 3 cm e se si misura una distanza inferiore ai 15 cm il sensore restituisce dei valori molto imprecisi. Proprio per questi motivi i risultati dell'esperimento possono variare di volta in volta e per ridurre questi effetti ho scelto come target le distanze 18, 25 e 30 centimetri ed ho riportato i valori medi ottenuti da 5 prove dell'esperimento per ciascun target.

Per eseguire questo esperimento ho sviluppato il seguente codice:

```
//FUNZIONE CHE ESEGUE IL TEST DI STABILITA'
//power : potenza settata per tutti i servomotori tra 0 e
   100
//target : distanza in cm fissata nell'esperimento
function test ( power , target ){
    var pw = (127 / 100) * power;
    echo ("pw : "+pw);
    var pwn = 255 - pw ;
    var i=0;

    at (sonar.val == target)
       stopMotor( 255, true , BRAKE ) & echo("Target
          Raggiunto in "+i+" passi") &
       stop forward & stop backward ,
    forward:at (sonar.val < target)
          setMotorOn( 255 , pwn , 0 , MOTORON ,
             REGULATION_MODE_IDLE , MOTOR_RUN_STATE_RUNNING
              ) & i++,
    backward:at (sonar.val > target)
          setMotorOn ( 255 , pw , 0 ,MOTORON ,
             REGULATION_MODE_IDLE , MOTOR_RUN_STATE_RUNNING
              ) & i++,

};
```

I risultati ottenuti in questo esperimento sono riportati nella tabella seguente, nella quale si riporta sulla prima colonna i valori della potenza in percentuale impostati e sulle tre colonne successive il numero di inversioni di marcia che il robot ripete prima di arrestarsi sui tre target fissati a 18, 25 e 30 cm. Se con la potenza fissata non si raggiunge mai il target, sulla tabella si riporta il simbolo ∞.

Queste prove sono avvenute dopo aver caricato completamente la batteria, di conseguenza ripetendo il test con una batteria più scarica i risultati possono essere molto differenti. Come si può facilmente notare sono stati riportati i risultati con potenza che parte dal 45% poichè per valori inferiori

Tabella 6.1: Risultati dell'esperimento

Power	Target = 18cm	Target = 25cm	Target = 30cm
45	1	1	1
50	3	4	3
55	8	5	7
60	∞	12	15
65	∞	∞	25
70	∞	∞	∞
75	∞	∞	∞
80	∞	∞	∞
85	∞	∞	∞
90	∞	∞	∞
95	∞	∞	∞
100	∞	∞	∞

a questo i servomotori non hanno la forza sufficiente a far muovere il robot. Nell'intervallo che va dal 45% al 55% il sistema risulta essere completamente stabile mentre man mano che si auemta la potenza oltre questi valori, a secondo del target scelto, a volte il sistema diventa instabile ed il robot continua a muoversi avanti ed indietro indefinitamente. Inoltre si può osservare che con una potenza impostata superiore al 70% della potenza disponibile il sistema è sempre instabile. Questo avviene perchè i ritardi che URBI introduce durante le operazioni di lettura del sensore e modifica dei parametri dei servomotori, sono superiori al tempo impiegato dal robot a raggiungere il target. Di conseguenza il sistema non riesce mai a fermare il robot alla distanza scelta.

In conclusione il sistema risulta essere stabile per valori di potenza compresi tra il 45% ed il 60% della potenza massima ed instabile per valori superiori al 60% della potenza massima.

Capitolo 7

Interfacciamento del robot WowWee Roboreptile con la piattaforma URBI

7.1 Introduzione

Il robot WowWee Roboreptile è comandato da un telecomando ad infrarossi. Per controllare questo robot trmmite PC è stato neccessario trovare un dispositivo che iniviasse segnali infrarossi alla frequenza adeguata: ho provato ad utilizzare dei normali adattatori infrarossi usb come quello in figura 7.1, ma questi dispositivi non lavorano alla frequenza di 39.2 KHz, necessaria al funzionamento della comunicazione. Dopo molte ricerche in rete ho sco-

Figura 7.1: Adattatore infrarossi USB

perto la possibilità di utilizzare la torre infrarossi della Lego per creare le segnalazioni dei comandi. La torre della Lego veniva adottata un tempo per programmare il predecessore del NXT chiamato RCX, dato che quest'ultimo aveva solamente interfaccia seriale. Durante questa ricerca ho torvato inoltre

Figura 7.2: Torre infrarossi Lego

un applicativo che utilizza la torre per comandare il robot WowWee Robosapien. Modificando il codice di questo applicativo, scritto in C++, sono riuscito a far muovere il Roboreptile comandandolo direttamente da prompt dei comandi. Una volta ottenuto questo risultato ho dovuto analizzare la documentazione degli UObject.

Gli UObject sono oggetti C++, che possono essere aggiunti alla piattaforma URBI per interfacciare nuovi dispositivi o per aggiungere funzionalità per i robot già interfacciabili. Le API per la creazione degli UObject definiscono la classe UObject. Ogni istanza di una classe derivata da questa, condivide tutti i metodi e gli attributi della classe UObject. Le API definiscono quali metodi e attributi della classe derivata siano condivisi con la classe UObject. Ad esempio per condividere una varibile con URBI si deve definire una UVar che rappresenta la variabile in questione. Questo tipo è un contenitore che fornisce operatori di casting e ugualianza per tutti i tipi di dato disponibili in URBI: double, string e char∗, e le strutture binarie UBinary, USound e UImage.
Le API forniscono metodi per la creazione di funzioni di callback che notificano quando una variabile viene modificata o letta da altro codice URBI. Per creare un UObject bisogna seguire i seguenti punti:

- includere le librerire UObject con il comando
  # include <urbi / uobject .hh >

- definire la classe dell'oggetto come erede da urbi::UObject.

- definire un unico costruttore avente come parametro una stringa e passare questa stringa al costruttore di urbi::UObject

- dichiare le varibili che si vogliono condividere con URBI usando il tipo urbi::UVar
- dichiare tutti i metodi della classe come funzioni
- nel costruttore usare le macro UBindVar(class-name,varible-name) per ogni UVar che si vuole istanziare e UBindFunction(class-name,function-name) per ogni funzione che si vuole collegare a URBI.
- richiamre la macro UStart alla fine di ogni oggetto

7.2 Codice per l'interfacciamento del robot con URBI

Per interfacciare il robot con URBI trammite la torre infrarossi Lego ho creato un file header C++ e la relativa classe C++ contenente la struttura e le funzioni dell'UObject. Il file header, riportato qui sotto, descrive quali varibili e quali funzioni devono essere condivise da questa classe con URBI.

```
# include <urbi/uobject.hh>

class roboreptile : public urbi :: UObject // Must inherit
    from UObject .
{
public :
// The class must have a single constructor taking a string
    .
roboreptile ( const std :: string &);
// Our variable .
urbi::UVar port;
// Our method .
void setPort(char *outport);

void up (int mode) ;

void down (int mode);

void left (int mode) ;

void right (int mode) ;

void stop (int mode) ;
```

```
void headLeft (int mode) ;

void headRight (int mode) ;

void demo (int mode) ;

void roam (int mode) ;

void feed (int mode) ;

};
```

Nella classe roboreptile.cpp invece, dopo aver importato le opportune librerie, sia quelle per l'ambiente Windows che quelle per l'ambiente Linux, ho dichiarato il costruttore che oltre a richiamare il costruttore della classe UObject, effettua il binding delle funzioni che volevo condividere con la piattaforma URBI. Poi ho definito le funzioni per utilizzare la torre infrarossi: la funzione append_bits accoda alcuni bit ad un comando per rendere tutti i comandi della stessa lunghezza, la funzione sap_open apre una connessione seriale con la torre infrarossi dei Lego, la funzione sap_send ha il compito di inviare un determinato segnale e la funzione sap_close chiude la connessione seriale con la torre dei Lego. Le altre funzioni hanno invece il compito di inviare i segnali in base al comando selezionato. Il codice seguente descrive il file roboreptile.cpp:

```
#ifdef WIN32
#include <windows.h>
#else
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <termios.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#define HANDLE int
#define INVALID_HANDLE_VALUE -1
#endif

# include "stdio.h"
# include "roboreptile.h"
```

```
// Costruttore della classe roboreptile
roboreptile :: roboreptile ( const std :: string & s)
: UObject (s) // required
{
   // Bind delle variabili
   UBindVar ( roboreptile , port );
   // Bind delle funzioni
   UBindFunction ( roboreptile , up );
   UBindFunction ( roboreptile , down );
   UBindFunction ( roboreptile , left );
   UBindFunction ( roboreptile , right );
   UBindFunction ( roboreptile , stop );
   UBindFunction ( roboreptile , headLeft );
   UBindFunction ( roboreptile , headRight);
   UBindFunction ( roboreptile , demo );
   UBindFunction ( roboreptile , roam );
   UBindFunction ( roboreptile , feed );
}

//Funzione che appende un bit in coda ad un comando
void append_bits(unsigned char *code, int *i, unsigned char
    *bitsfilled, int n, unsigned char bit)
{
  bit <<= 7;
  /* first fill the current byte */
  while (*bitsfilled < 10)
  {
    if ((*bitsfilled > 0) && (*bitsfilled < 9))
    {
      code[*i] >>= 1;
      code[*i] |= bit;
    }
    (*bitsfilled)++; n--;
  }

  /* now fill all "full" bytes */
  while (n >= 10)
  {
    code[++(*i)] = bit?255:0;
    n -= 10;
  }

  /* and the last few bits that start another byte */
```

```
  (*i)++; *bitsfilled = 0;
  while (n > 0)
  {
    if ((*bitsfilled > 0) && (*bitsfilled < 9))
    {
      code[*i] >>= 1;
      code[*i] |= bit;
    }
    (*bitsfilled)++; n--;
  }
}

#ifndef WIN32
struct termios oldtio;
#endif

//Funzione che apre un handle con la porta COM
HANDLE sap_open(const char *outport)
{
  HANDLE f;
#ifdef WIN32
  f = CreateFile(outport, GENERIC_READ | GENERIC_WRITE, 0,
     NULL, OPEN_EXISTING, 0, NULL);
  if (f == INVALID_HANDLE_VALUE)
  {
       printf("sapien: error opening port\n");
    return f;
  }

  if (!SetupComm(f, 64, 1024))
  {
    printf("sapien: Cannot setup port buffers\n");
    CloseHandle(f);
    return INVALID_HANDLE_VALUE;
  }

  DCB dcb;
  FillMemory(&dcb, sizeof(dcb), 0);

  if (!GetCommState(f, &dcb))
  {
    printf("sapien: GetCommState() error: %d\n",
       GetLastError());
```

```
    CloseHandle(f);
    return INVALID_HANDLE_VALUE;
  }

  dcb.BaudRate = CBR_115200;
  dcb.fParity = FALSE;
  dcb.fOutxCtsFlow = FALSE;
  dcb.fOutxDsrFlow = FALSE;
  dcb.fDtrControl = DTR_CONTROL_DISABLE;
  dcb.fDsrSensitivity = FALSE;
  dcb.fTXContinueOnXoff = TRUE;
  dcb.fOutX = FALSE;
  dcb.fInX = FALSE;
  dcb.fRtsControl = RTS_CONTROL_DISABLE;
  dcb.fAbortOnError = FALSE;
  dcb.ByteSize = 8;
  dcb.Parity = NOPARITY;
  dcb.StopBits = ONESTOPBIT;

  if (!SetCommState(f, &dcb))
  {
    printf("sapien: SetCommState() error %d\n",
       GetLastError());
    CloseHandle(f);
    return INVALID_HANDLE_VALUE;
  }

#else
  f = open(outport, O_RDWR | O_NOCTTY | O_NONBLOCK);
  if (f < 0)
  {
    perror(outport);
    return INVALID_HANDLE_VALUE;
  }

  struct termios newtio;
  //int fd = open(outport, O_RDWR | O_NOCTTY | O_NONBLOCK);
  //if (fd < 0) { perror(outport); return
     INVALID_HANDLE_VALUE; }

  //tcgetattr(fd,&oldtio); /* save current port settings */
  tcgetattr(f,&oldtio); /* save current port settings */
```

```
  bzero(&newtio, sizeof(newtio));
  newtio.c_cflag = B115200 | CS8 | CLOCAL ;
  newtio.c_iflag = IGNPAR;
  newtio.c_oflag = 0;
  newtio.c_lflag = 0;
  newtio.c_cc[VTIME] = 0;
  newtio.c_cc[VMIN] = 1;
  cfsetspeed(&newtio, B115200);
  //tcflush(fd, TCIFLUSH);
  tcflush(f, TCIFLUSH);
  // if (tcsetattr(fd,TCSANOW,&newtio))
  if (tcsetattr(f,TCSANOW,&newtio))
  {
    perror("tcsetattr()");
    //close(fd);
    close(f);
    return INVALID_HANDLE_VALUE;
  }
#endif

  return f;
}

//Funzione che chiude un handle
void sap_close(HANDLE f)
{
#ifdef WIN32
  CloseHandle(f);
#else
  tcsetattr(f,TCSANOW,&oldtio);
  close(f);
#endif
}

//Funzione che invia un segnale infrarosso

void sap_send(HANDLE f, int nowakeup, int b)
{
  int x = (b < 0x100)?128:2048;
  if (b>4095) x = 2097152;
  unsigned char code[1024];
  int i;
```

```
  unsigned char bitsfilled = 0;  /* 0-10  (counting also
     start and stop bits) */

  if (!nowakeup)
  {
      char *buf = "\0x00";
#ifdef WIN32
      DWORD written;
      WriteFile(f, buf, 1, &written, 0);
      FlushFileBuffers(f);
      Sleep(100);
#else
      write(f, buf, 1);
      struct timespec tm;
      tm.tv_sec = 0;
      tm.tv_nsec = 100000000;
      nanosleep(&tm, 0);
#endif
  }

  code[0] = 0;
  int ps = 4;
  int tm = 96;
  int preamble = 8;

  if (b > 4095)
  {
          preamble = 5;
          ps = 2;
          tm = 64;
  }
  i = 0;
  append_bits(code, &i, &bitsfilled, tm * preamble, 0);

  while (x)
  {
    if (b & x) append_bits(code, &i, &bitsfilled, tm * ps,
       1);
    else append_bits(code, &i, &bitsfilled, tm, 1);
    append_bits(code, &i, &bitsfilled, tm, 0);
    x >>= 1;
  }
```

```
  if (bitsfilled > 0) i++;
  unsigned long written;
  int w = 0;
  while (w < i)
  {
#ifdef WIN32
    if (0 == WriteFile(f, code + w, i, &written, 0))
    {
      printf("sapien: problems writing to file, %d\n",
         GetLastError());
      return;
    }
#else
    written = write(f, code + w, i);
    if (written == -1)
    {
      printf("sapien: problems writing to file, %d\n",
         errno);
      return;
    }
#endif
    w += (int)written;
  }
}

void roboreptile :: up (int mode)
{
    int nowakeup = 0 ;
    #ifdef WIN32
    char *outport = "COM1" ;
    #else
    char *outport = "/dev/ttyS0" ;
    #endif
    roboreptile::port = outport ;
    int version = 1 ;
    int code;
    if (mode=0){
       code = 0x481 ;
    }else if (mode=1){
       code = 0x491 ;
    }else{
       code = 0x4A1 ;
    }
```

```
    printf("code: "+code);
    if ((code > 0xFF) && (version == 1))
        version = 2;
    HANDLE f = sap_open(outport);
    //if (f == INVALID_HANDLE_VALUE)
      //errore
      //printf("Errore....");
      //return 1 ;
    sap_send(f, 0, code);
    sap_close(f);
    //return 0 ;
}

void roboreptile :: down (int mode)
{
    int nowakeup = 0 ;
    #ifdef WIN32
    char *outport = "COM1" ;
    #else
    char *outport = "/dev/ttyS0" ;
    #endif
    roboreptile::port = outport ;
    int version = 1 ;
    //int code = 0x482 ;
    int code;
    if (mode=0){
        code = 0x482 ;
    }else if (mode=1){
        code = 0x492 ;
    }else{
        code = 0x4A2 ;
    }
    if ((code > 0xFF) && (version == 1))
        version = 2;
    HANDLE f = sap_open(outport);
    if (f == INVALID_HANDLE_VALUE)
      //errore
      printf("Errore....");
      //return 1 ;
    sap_send(f, 0, code);
    sap_close(f);
    //return 0 ;
}
```

```
void roboreptile :: left (int mode)
{
    int nowakeup = 0 ;
    #ifdef WIN32
    char *outport = "COM1" ;
    #else
    char *outport = "/dev/ttyS0" ;
    #endif
    roboreptile::port = outport ;
    int version = 1 ;
    //int code = 0x483 ;
    int code;
    if (mode=0){
       code = 0x483 ;
    }else if (mode=1){
       code = 0x493 ;
    }else{
       code = 0x4A3 ;
    }
    if ((code > 0xFF) && (version == 1))
       version = 2;
    HANDLE f = sap_open(outport);
    if (f == INVALID_HANDLE_VALUE)
      //errore
      printf("Errore....");
      //return 1 ;
    sap_send(f, 0, code);
    sap_close(f);
    //return 0 ;
}

void roboreptile :: right (int mode)
{
    int nowakeup = 0 ;
    #ifdef WIN32
    char *outport = "COM1" ;
    #else
    char *outport = "/dev/ttyS0" ;
    #endif
    roboreptile::port = outport ;
    int version = 1 ;
    //int code = 0x484 ;
```

```
    int code;
    if (mode=0){
       code = 0x484 ;
    }else if (mode=1){
       code = 0x494 ;
    }else{
       code = 0x4A4 ;
    }
    if ((code > 0xFF) && (version == 1))
       version = 2;
    HANDLE f = sap_open(outport);
    if (f == INVALID_HANDLE_VALUE)
      //errore
      printf("Errore....");
      //return 1 ;
    sap_send(f, 0, code);
    sap_close(f);
    //return 0 ;
}

void roboreptile :: stop (int mode)
{
    int nowakeup = 0 ;
    #ifdef WIN32
    char *outport = "COM1" ;
    #else
    char *outport = "/dev/ttyS0" ;
    #endif
    roboreptile::port = outport ;
    int version = 1 ;
    //int code = 0x485 ;
    int code;
    if (mode=0){
       code = 0x485 ;
    }else if (mode=1){
       code = 0x495 ;
    }else{
       code = 0x4A5 ;
    }
    if ((code > 0xFF) && (version == 1))
       version = 2;
    HANDLE f = sap_open(outport);
    if (f == INVALID_HANDLE_VALUE)
```

```
      //errore
      printf("Errore....");
      //return 1 ;
    sap_send(f, 0, code);
    sap_close(f);
    //return 0 ;
}

void roboreptile :: headLeft (int mode)
{
    int nowakeup = 0 ;
    #ifdef WIN32
    char *outport = "COM1" ;
    #else
    char *outport = "/dev/ttyS0" ;
    #endif
    roboreptile::port = outport ;
    int version = 1 ;
    //int code = 0x486 ;
    int code;
    if (mode=0){
       code = 0x486 ;
    }else if (mode=1){
       code = 0x496 ;
    }else{
       code = 0x4A6 ;
    }
    if ((code > 0xFF) && (version == 1))
       version = 2;
    HANDLE f = sap_open(outport);
    if (f == INVALID_HANDLE_VALUE)
      //errore
      printf("Errore....");
      //return 1 ;
    sap_send(f, 0, code);
    sap_close(f);
    //return 0 ;
}

void roboreptile :: headRight (int mode)
{
    int nowakeup = 0 ;
    #ifdef WIN32
```

```
    char *outport = "COM1" ;
    #else
    char *outport = "/dev/ttyS0" ;
    #endif
    roboreptile::port = outport ;
    int version = 1 ;
    //int code = 0x487 ;
    int code;
    if (mode=0){
       code = 0x487 ;
    }else if (mode=1){
       code = 0x497 ;
    }else{
       code = 0x4A7 ;
    }
    if ((code > 0xFF) && (version == 1))
       version = 2;
    HANDLE f = sap_open(outport);
    if (f == INVALID_HANDLE_VALUE)
      //errore
      printf("Errore....");
      //return 1 ;
    sap_send(f, 0, code);
    sap_close(f);
    //return 0 ;
}

void roboreptile :: demo (int mode)
{
    int nowakeup = 0 ;
    #ifdef WIN32
    char *outport = "COM1" ;
    #else
    char *outport = "/dev/ttyS0" ;
    #endif
    roboreptile::port = outport ;
    int version = 1 ;
    //int code = 0x488 ;
    int code;
    if (mode=0){
       code = 0x488 ;
    }else if (mode=1){
       code = 0x498 ;
```

```
    }else{
       code = 0x4A8 ;
    }
    if ((code > 0xFF) && (version == 1))
       version = 2;
    HANDLE f = sap_open(outport);
    if (f == INVALID_HANDLE_VALUE)
      //errore
      printf("Errore....");
      //return 1 ;
    sap_send(f, 0, code);
    sap_close(f);
    //return 0 ;
}

void roboreptile :: roam (int mode)
{
    int nowakeup = 0 ;
    #ifdef WIN32
    char *outport = "COM1" ;
    #else
    char *outport = "/dev/ttyS0" ;
    #endif
    roboreptile::port = outport ;
    int version = 1 ;
    //int code = 0x489 ;
    int code;
    if (mode=0){
       code = 0x489 ;
    }else if (mode=1){
       code = 0x499 ;
    }else{
       code = 0x4A9 ;
    }
    if ((code > 0xFF) && (version == 1))
       version = 2;
    HANDLE f = sap_open(outport);
    if (f == INVALID_HANDLE_VALUE)
      //errore
      printf("Errore....");
      //return 1 ;
    sap_send(f, 0, code);
    sap_close(f);
```

```
    //return 0 ;
}

void roboreptile ::feed (int mode)
{
    printf("Invio il comando UP");
    int nowakeup = 0 ;
    #ifdef WIN32
    char *outport = "COM1" ;
    #else
    char *outport = "/dev/ttyS0" ;
    #endif
    roboreptile::port = outport ;
    int version = 1 ;
    //int code = 0x480 ;
    int code;
    if (mode=0){
       code = 0x480 ;
    }else if (mode=1){
       code = 0x490 ;
    }else{
       code = 0x4A0 ;
    }
    if ((code > 0xFF) && (version == 1))
       version = 2;
    HANDLE f = sap_open(outport);
    if (f == INVALID_HANDLE_VALUE)
      //errore
      printf("Errore....");
      //return 1 ;
    sap_send(f, 0, code);
    sap_close(f);
    //return 0 ;
}

UStart ( roboreptile );
```

Ho sviluppato l'UObject roboreptile in modo che sia portabile sia su sistemi operativi Microsoft Windows, sia su sistemi operativi Linux.

Dopo aver sviluppato il codice descritto precedentemente sono passato alla fase di compilazione. Durante questa fase, con il sistema operativo Linux, la compilazione è andata a buon fine ed è stato possibile provare il funzionamento del robot. Il robot, comandato dal PC trammite URBI e torre dei

Lego funziona come se venisse comandato dal suo telecomando. In ambiente Microsoft Windows invece ho avuto dei problemi in compilazione legati alle librerie dll. Infatti le librerie dll della distribuzione urbi-sdk-2.0 non sono compatibili con il compilatore mingw. Chiedendo sul forum di supporto di URBI 2.0 ho saputo che è necessario compilare gli UObject in ambiente Windows con Microsoft Visual Studio ma anche provando a compilare in tale piattaforma non sono riuscito ad ottenere un modulo corretto da collegare al Urbi-server. Di conseguenza non è stato possibile provare il robot in ambiente Windows.

7.3 Esperimento: Uso del linguaggio URBI con i robot NXT e Roboreptile

Per testare il sistema ho cercato di confrontare il comportamento del Roboreptile e del NXT. Per far questo ho creato un set di movimenti uguali per entrambi i robot. Dato che il Roboreptile ha dei comandi più restrittivi rispetto al NXT, ho deciso di creare delle funzioni per il robot NXT che simulino i movimenti del Roboreptile. Queste funzioni sono riportate nel file movement.u, il cui contenuto è il seguente:

```
//FUNZIONE CHE FA MUOVERE IN AVANTI IL ROBOT CON UNA
   VELOCITA' SPECIFICATA
//spd: velocita espressa in % della velocità massima,
   valore compreso tra 0 e 100;
function forward(spd){
    if ( spd>=0 & spd<=100)
        wheels.speed=spd;
}
//FUNZIONE CHE FA MUOVERE ALL'INDIETRO IL ROBOT CON UNA
   VELOCITA' SPECIFICATA
//spd: velocita espressa in % della velocità massima,
   valore compreso tra 0 e 100;
function backward(spd){
    if (spd>=0 & spd<=100)
        wheels.speed=-spd;
}
//FUNZIONE CHE FA RUOTARE VERSO SINISTRA IL ROBOT CON UNA
   VELOCITA' SPECIFICATA
//spd: velocita espressa in % della velocità massima,
   valore compreso tra 0 e 100;
function left(spd){
    if (spd>=0 & spd<=100){
```

```
        wheelL.speed=spd;
        wheelR.speed=-spd;
    }
}
//FUNZIONE CHE FA RUOTARE VERSO DESTRA IL ROBOT CON UNA
   VELOCITA' SPECIFICATA
//spd: velocita espressa in % della velocità massima,
   valore compreso tra 0 e 100;
function right(spd){
    if (spd>=0 & spd<=100){
        wheelR.speed=spd;
        wheelL.speed=-spd;
    }
}
//FUNZIONE CHE ARRESTA IL ROBOT
function stop(){
    wheels.speed=0;
}

//FUNZIONE CHE FA MUOVERE IN AVANTI IL ROBOT
function forward(){
    wheels.speed=45;
}
//FUNZIONE CHE FA MUOVERE ALL'INDIETRO IL ROBOT
//spd: velocita espressa in % della velocità massima,
   valore compreso tra 0 e 100;
function backward(){
        wheels.speed=-45;
}
//FUNZIONE CHE FA RUOTARE VERSO SINISTRA IL ROBOT
//spd: velocita espressa in % della velocità massima,
   valore compreso tra 0 e 100;
function left(){
        wheelL.speed=45;
        wheelR.speed=-45;
}
//FUNZIONE CHE FA RUOTARE VERSO DESTRA IL ROBOT
//spd: velocita espressa in % della velocità massima,
   valore compreso tra 0 e 100;
function right(){
        wheelR.speed=45;
        wheelL.speed=-45;
}
```

Si riporta una tabella con le corrispondenze tra le funzioni descritte precedentemente ed i comandi del roboreptile:

Tabella 7.1: Corrispondenze delle funzioni

Funzione NXT	Funzione Roboreptile
forward()	up(0)
backward()	down(0)
left()	left(0)
right()	right(0)
stop()	stop(0)

Una volta che ho ottenuto dei layout dei comandi uguali per i due robot, ho fatto eseguire una serie di movimenti identici per entrambi. I comandi che ho fatto eseguire sono i seguenti:

Tabella 7.2: Comandi inviati ai robot

NXT	Roboreptile
forward();	uobjects.roboreptile.up(0);
wait(15s);	wait(15s);
stop();	uobjects.roboreptile.stop(0);
left();	uobjects.roboreptile.left(0);
wait(5s);	wait(5s);
stop();	uobjects.roboreptile.stop(0);
forward();	uobjects.roboreptile.up(0);
wait(15s);	wait(15s);
stop();	uobjects.roboreptile.stop(0);
backward();	uobjects.roboreptile.down(0);
wait(15s);	wait(15s);
stop();	uobjects.roboreptile.stop(0);
right();	uobjects.roboreptile.right(0);
wait(5s);	wait(5s);
stop();	uobjects.roboreptile.stop(0);
backward();	uobjects.roboreptile.down(0);
wait(15);	wait(15);
stop();	uobjects.roboreptile.stop(0);

In conclusione i robot si comportano in modo molto simile anche se il Lego Mindstorm NXT risulta essere molto più veloce ed agile rispetto al Roboreptile. Questo è dovuto alla struttura dei due robot. Comunque ho dimostrato, in questo modo, la flesibilità della piattaforma URBI. Ho dimostrato anche la portabilità della piattaforma, dato che un robot viene comandato in ambiente Microsoft Windows e l'altro in ambiente Linux.

Manuale Utente

Manuale Utente dell'interfaccia per i sensori opzionali e i motori del robot Lego Mindstorms NXT

Le funzioni per la lettura dei dati dei sensori opzionali e quelle per pilotare i motori del robot didattico Lego Mindstorms NXT sono raccolte un in file chiamato controll.u. Dopo aver avviato il server ed il client e aver caricato il file controll.u si possono interfacciare i sensori seguendo le indicazioni seguenti.

Per leggere le accelerazioni lungo i tre assi x, y, z, rilevate dall'accelerometro (Tilt/Acellerometer Sensor), si devono usare le funzioni seguenti nell'ordine con cui sono riportate in questo documento:

```
setInputMode( porta , LOWSPEED9V , RAW );
setUSMode ( porta , US_MODE_CONTINUOUS );
var v = getAccelerometerVal ( PORT4 );
var x = v[3];
var y = v[4];
var z = v[5];
if (x>127) x-=256
x=x*4+v[6]
if (y>127) y-=256
y=y*4+v[7]
if (z>127) z-=256
z=z*4+v[8]
```

Invece, per leggere le accellerazioni solo lungo l'asse x,rilevate dal giroscopio (Gyro Sensor), si devono usare le funzioni riportate nell'ordine seguente.

```
setInputMode( PORT3 , NO_SENSOR , RAW );
var v = getAnalogicSensorVal ( PORT3 )
var byte7 = v[7];
```

```
var byte8 =v[8];
var mis=(byte7*4)+byte8;
```

Per ottenere il colorNumber rilevato dal sensore di colori (Color Sensor) bisogna usare le funzioni nell'ordine seguente:

```
setInputMode( porta , LOWSPEED9V , RAW );
setUSMode ( porta , US_MODE_CONTINUOUS );
var v = getI2CSensorVal ( porta );
var ColorNumber = v[3];
```

Per conoscere invece i gradi misurati dal sensore bussola (Compass Sensor) si devono inviare i seguenti comandi:

```
setInputMode( porta , LOWSPEED9V , RAW );
setUSMode ( porta , US_MODE_CONTINUOUS );
var v = getI2CSensorVal ( porta );
var degrees = v[3];
```

Manuale Utente dell'interfaccia per il controllo del robot WowWee Roboreptile

Per controllare il Roboreptile trammite la piattaforma urbi in ambiente Linux è necessario inanzitutto creare i path con i comandi seguenti comandi:

```
URBI_ROOT="perorso in cui si trova la cartella urbi-sdk-2.0
    "
LD_LIBRARY_PATH="$URBI_ROOT/gostai/core/i686-pc-linux-gnu/
    engine:$LD_LIBRARY_PATH"
LD_LIBRARY_PATH="$URBI_ROOT/lib:$LD_LIBRARY_PATH"
export LD_LIBRARY_PATH
```

E' necessario anche impostare la velocità della porta seriale a 115200 baud con il comando:

```
stty -F /dev/ttyS0 115200
```

Dopodichè è possibile avviare il server sulla porta 54000 facendo caricare il modulo contente l'UObject per il controllo del robot con il comando:

```
$URBI_ROOT/bin/urbi-launch --start --port 54000 ./
    roboreptile.so
```

Una volta avviato il server, si possono usare vari terminali per connettersi. In questa guida si riporta il comando per utilizzare un terminale telnet dalla stessa macchina:

```
telnet 127.0.0.1 54000
```

Altrimenti, se si vuole aprire il terminale su un altra macchina basta inviare il comando precedente modificando l'indirizzo IP 127.0.0.1 con quello della macchina sulla quale si fa eseguire il server.

Ora che si è aperto il terminale è possibile inviare i comandi per pilotare il robot. Per far muovere in avanti il roboreptile si deve inviare il comando:

```
uobjects.roboreptile.up(0);
```

Mentre per farlo muovere all'indietro si deve digitare:

```
uobjects.roboreptile.down(0);
```

Se si vuole far ruotare il robot a sinistra o a destra , si devono inviare rispettivamente i comandi:

```
uobjects.roboreptile.left(0);
```

```
uobjects.roboreptile.right(0);
```

Invece per arrestare qualsiasi movimento del robot si può digitare uno dei seguenti comandi:

```
uobjects.roboreptile.stop(0);
```

```
uobjects.roboreptile.stop(1);
```

```
uobjects.roboreptile.stop(2);
```

Per far muovere la testa del robot rispettivamente a sinistra e a destra si devono inviare i comandi:

```
uobjects.roboreptile.headLeft(0);
```

```
uobjects.roboreptile.headRight(0);
```

Per far alzare il robot su due zampe o viceversa farlo abbassare su quattro zampe si possono usare, rispettivamente, i comandi:

```
uobjects.roboreptile.up(1);
uobjects.roboreptile.down(1);
```

E' anche possibile far mangiare il robot con uno dei seguenti comandi:

```
uobjects.roboreptile.feed(0);
```

```
uobjects.roboreptile.feed(1);
```

```
uobjects.roboreptile.feed(2);
```

Oppure si può farlo attacare o squotersi rispettivamente con i comandi:

```
uobjects.roboreptile.headLeft(2);
uobjects.roboreptile.headRight(2);
```

Opuure si può far saltare il robot con il comando:

```
uobjects.roboreptile.up(2);
```

o farlo perlustrare la zona con:

```
uobjects.roboreptile.down(2);
```

Il robot può inoltre eseguire delle demo dei suoi movimenti. Per ottenere questo si inviano i comandi:

```
uobjects.roboreptile.demo(0);
```

```
uobjects.roboreptile.demo(1);
```

```
uobjects.roboreptile.demo(2);
```

Infine ci sono tre comandi che modificano il comportamento del robot: il primo ordina al robot di vagabondare in cerca di interazioni con l'ambiente che lo circonda, il secondo imposta la modalità guardia nella quale il robot rimane in attesa di stimoli esterni (rumori e movimenti) ed il terzo rende il robot aggressivo verso qualunque cosa:

```
uobjects.roboreptile.roam(0);
```

```
uobjects.roboreptile.roam(1);
```

```
uobjects.roboreptile.roam(2);
```

Esiste anche la possibilità di far salvare una serie di comandi al robot e farli rieseguire in serie in un secondo momento. Per far questo si deve inviare il comando:

```
uobjects.roboreptile.headLeft(1);
```

che imposta la modalità programma , poi inviare i comandi da salvare ed infine inviare il comando per l'esecuzione del programma cioè:

```
uobjects.roboreptile.headRight(1);
```

Quando si vuole ripetere la sequenza di operazioni basta inviare il comando precedente.

In questo modo si riesce a controllare qualsiasi movimento e comportamento del Roboreptile.

Manuale Tecnico

Manuale Tecnico dell'interfaccia per il controllo del robot WowWee Roboreptile

Per sviluppare l'interfaccia per il controllo del Roboreptile è necessario conoscere le funzioni che il sistema operativo mette a disposizione per inviare dei segnali attraverso le porte seriali poichè il dispositivo usato per collegare il robot al PC è di tipo seriale. Di conseguenza è necessario scrivere del codice che si adatti al sistema operativo in uso. In C++ questo è reso possibile con la sintassi

```
#ifdef WIN32
"codice per s.o. windows"
#else
"codice per s.o. linux"
#endif
```

Infatti ho sviluppato l'UObject roboreptile in modo che sia compilabile ed eseguibile in entrambe le famigli di sistemi operativi. Per utilizzare le porte seriali in Windows è necessario importare la libreria *windows.h* mentre in ambienti linux sono necessarie tutte le seguenti librerie: *unistd.h*, *string.h*, *time.h*, *termios.h*, *sys/types.h*, *sys/stat.h*, *fcntl.h* e *errno.h*. Inoltre, per entrambi i sistemi operativi è necessario importare la libreria *stdio.h* che fornisce lo standard output.

Vediamo ora come aprire una connessione con il dispositivo collegato alla porta seriale:tutto il codice seguente è contenuto nella funzione *sap_open* che crea la connessione con un dispositivo collegato con la porta seriale specificata.
Nel caso ci si trovi in ambiente Windows si deve creare un file con la riga di codice seguente:

```
f = CreateFile(outport, GENERIC_READ | GENERIC_WRITE, 0,
    NULL, OPEN_EXISTING, 0, NULL);
```

dove outport è la porta COM passata come parametro sulla quale è connessa la torre Lego. Questa funzione restituisce un intero che rappresenta l'idetificativo della connessione. Se tale intero, chiamato *HANDLE* è uguale a -1 significa che c'è stato un errore e la comunicazione non va a termine. Se non avvengono errori si deve impostare la comunicazione e per farlo si usa il codice seguente, nella quale viene effettuato contemporaneamente un controllo:

```
if (!SetupComm(f, 64, 1024))
  {
    printf("sapien: Cannot setup port buffers\n");
    CloseHandle(f);
    return INVALID_HANDLE_VALUE;
  }
```

Dopodichè, si crea un oggetto *dcb* che serve a memorizzare le impostazioni della porta seriale e lo si fa riempire con il comando:

```
  FillMemory(&dcb, sizeof(dcb), 0);
```

A questo punto, con il codice seguente, si richiedono le impostazioni della connessione e le si salvano nell'oggetto dcb.

```
if (!GetCommState(f, &dcb))
  {
    printf("sapien: GetCommState() error: %d\n",
       GetLastError());
    CloseHandle(f);
    return INVALID_HANDLE_VALUE;
  }
```

Ora non resta altro che impostare la porta con le impostazioni necessarie all'utilizzo della torre infrarossi:

```
dcb.BaudRate = CBR_115200;
  dcb.fParity = FALSE;
  dcb.fOutxCtsFlow = FALSE;
  dcb.fOutxDsrFlow = FALSE;
  dcb.fDtrControl = DTR_CONTROL_DISABLE;
  dcb.fDsrSensitivity = FALSE;
  dcb.fTXContinueOnXoff = TRUE;
  dcb.fOutX = FALSE;
  dcb.fInX = FALSE;
  dcb.fRtsControl = RTS_CONTROL_DISABLE;
  dcb.fAbortOnError = FALSE;
  dcb.ByteSize = 8;
  dcb.Parity = NOPARITY;
```

```
dcb.StopBits = ONESTOPBIT;
```

Infine basta richiamare la funzione *SetCommState* per impostare i parametri appena modificati:

```
if (!SetCommState(f, &dcb))
  {
    printf("sapien: SetCommState() error %d\n",
        GetLastError());
    CloseHandle(f);
    return INVALID_HANDLE_VALUE;
  }
```

Invece se il sistema operativo su cui si lavora è linux si deve aprire la connessione con il codice seguente:

```
 f = open(outport, O_RDWR | O_NOCTTY | O_NONBLOCK);
  if (f < 0)
  {
    perror(outport);
    return INVALID_HANDLE_VALUE;
  }
```

dove outport è la porta sulla quale la torre Lego è connessa. Dopodichè è necessario salvare lo stato della porta con il comando:

```
  tcgetattr(f,&oldtio);
```

in cui oldtio è una variabile globale accessibile anche dalle altre funzioni dell'UObject. Arrivati a questo punto si devono salvare le impostazioni della porta in un oggetto chiamato newtio con il codice seguente:

```
  struct termios newtio;
  bzero(&newtio, sizeof(newtio));
  newtio.c_cflag = B115200 | CS8 | CLOCAL ;
  newtio.c_iflag = IGNPAR;
  newtio.c_oflag = 0;
  newtio.c_lflag = 0;
  newtio.c_cc[VTIME] = 0;
  newtio.c_cc[VMIN] = 1;
  cfsetspeed(&newtio, B115200);
  tcflush(f, TCIFLUSH);
 if (tcsetattr(f,TCSANOW,&newtio))
  {
    perror("tcsetattr()");
    //close(fd);
    close(f);
```

```
    return INVALID_HANDLE_VALUE;
  }
```

Invece per inviare un segnale attraverso la porta si devono eseguire nell'ordine le seguenti operazioni: si deve adattare il segnale rappresentato con un intero b, operando delle operazioni di bit stuffing ed altre operazioni che modificano la lunghezza del comando:

```
  int x = (b < 0x100)?128:2048;
  if (b>4095) x = 2097152;
  unsigned char code[1024];
  int i;
  unsigned char bitsfilled = 0;  /* 0-10  (counting also
     start and stop bits) */

  if (!nowakeup)
  {
      char *buf = "\0x00";
#ifdef WIN32
      DWORD written;
      WriteFile(f, buf, 1, &written, 0);
      FlushFileBuffers(f);
      Sleep(100);
#else
      write(f, buf, 1);
      struct timespec tm;
      tm.tv_sec = 0;
      tm.tv_nsec = 100000000;
      nanosleep(&tm, 0);
#endif
  }

  code[0] = 0;
  int ps = 4;
  int tm = 96;
  int preamble = 8;

  if (b > 4095)
  {
          preamble = 5;
          ps = 2;
          tm = 64;
  }
  i = 0;
```

```
  append_bits(code, &i, &bitsfilled, tm * preamble, 0);

  while (x)
  {
    if (b & x) append_bits(code, &i, &bitsfilled, tm * ps,
       1);
    else append_bits(code, &i, &bitsfilled, tm, 1);
    append_bits(code, &i, &bitsfilled, tm, 0);
    x >>= 1;
  }
```

Dopodichè si invia il segnale ottenuto con il codice seguente:

```
if (bitsfilled > 0) i++;
  unsigned long written;
  int w = 0;
  while (w < i)
  {
#ifdef WIN32
    if (0 == WriteFile(f, code + w, i, &written, 0))
    {
      printf("sapien: problems writing to file, %d\n",
         GetLastError());
      return;
    }
#else
    written = write(f, code + w, i);
    if (written == -1)
    {
      printf("sapien: problems writing to file, %d\n",
         errno);
      return;
    }
#endif
    w += (int)written;
  }
```

Il codice illustrato precedentemente corrisponde alla funzione *sap_send* ha il compito di inviare un segnale attraverso la porta seriale selezionata.

Infine, per terminare la connessione e ripristinare lo stato della porta seriale, in ambiente Windows, si deve usare la seguente righa di codice:

```
CloseHandle(f);
```

Mentre in ambiente Linux questo è possibile con il codice:

```
  tcsetattr(f,TCSANOW,&oldtio);
```

```
close(f);
```

Queste righe di codice sono contenute nella funzione *sap-close* che ha il compito di chiudere la connessione con una porta seriale.

La funzione *append_bits* ha il compito di accodare dei bit in coda al segnale.

Per inviare un qualsiasi comando al Roboreptile si deve:

1. definire la porta seriale sulla quale la torre infrarossi è connessa: in windows sarà la porta *COM1* ed in Linux la porta */dev/ttyS0*

2. aprire la connessione con la porta seriale con il codice:
   HANDLE f = sap_open(outport);

3. inviare il segnale corrispondente al comando: *sap_send(f, 0, code);*

4. chiudere la connessione con la porta ripristinando il suo stato iniziale:
   sap_close(f);

Per esempio riportiamo la funzione che invia i comandi avanti, alzati su due zampe e salta a seconda del layer scelto con il parametro mode:

```
void roboreptile :: up (int mode)
{
    int nowakeup = 0 ;
    #ifdef WIN32
    char *outport = "COM1" ;
    #else
    char *outport = "/dev/ttyS0" ;
    #endif
    roboreptile::port = outport ;
    int version = 1 ;
    int code;
    if (mode=0){
        code = 0x481 ;
    }else if (mode=1){
        code = 0x491 ;
    }else{
        code = 0x4A1 ;
    }
    printf("code: "+code);
    if ((code > 0xFF) && (version == 1))
        version = 2;
    HANDLE f = sap_open(outport);
    //if (f == INVALID_HANDLE_VALUE)
```

```
        //errore
        //printf("Errore....");
        //return 1 ;
    sap_send(f, 0, code);
    sap_close(f);
    //return 0 ;
}
```

Conclusioni

In questa trattazione ho analizzato le caratteristiche della piattaforma universale per il controllo dei robot URBI. In particolare ho provato questa piattaforma con il robot Lego Mindstorms NXT per la quale esiste una specifica distribuzione. Poi ho sviluppato l'interfaccia che permette al robot di utilizzare i sensori opzionali. Durante questo sviluppo ho potuto provare la flessibilità e la semplicità del linguaggio di scripting urbiScript ed testato il funzionamento del protocollo dei comandi diretti per il robot NXT. Per esaminare questa piattaforma con il robot NXT ho creato un esperimento nella quale sono emerse delle problematiche di stabilità legate principalmente ai ritardi che il sistema nel suo complesso comporta. Questi ritardi, dovuti all'archittetura client-server e al protocollo dei comandi diretti del NXT, fanno si che, se le richieste avvengono con dei tempi troppo stringenti per il sistema, questo non riesce a soddisfarle diventando instabile. Bisogna quindi tener presente questo fatto quando si vuole controllare il robot con questa piattaforma.
E' stato inoltre possibile testare l'architettura client-server della piattaforma in quanto ho provato a comandare il robot Lego Mindstorms NXT collegato ad un PC da un altra macchina.
Un'altra analisi effettuata sulla piattaforma riguarda la sua portabilità su altri robot ed il suo utilizzo in sistemi operativi diversi quali Microsoft Windows e Linux Ubuntu. In particolare ho interfacciato completamente un robot che non aveva nessuna interfaccia con il PC. Durante questa parte dell'elaborato ho creato un oggetto embedded da collegare alla piattaforma che fornisce l'interfaccia di controllo del robot WowWee Roboreptile. Da questa analisi è emersa la facilità con cui si possono interfacciare robot ma anche altri dispositivi (in questo caso la torre infrarossi per inviare i comandi al robot) con la piattaforma.
In ambiente Linux la compilazione è andata subito a buon fine, sono riuscito ad ottenere un modulo con estensione .so da collegare al server URBI e ho potuto testare il funzionamento del Roboreptile in Linux.
In ambiente Microsoft Windows, invece, ho avuto dei problemi durante la

fase di compilazione dovuti alle librerie della distribuzione urbi-sdk-2.0, in quanto queste non erano compatibili con il compilatore open-surce mingw32. Ho provato quindi, come suggerito dagli sviluppatori di URBI, a compilare il mio oggetto embedded con Microsoft Visual Studio, ma anche in questo caso non ho ottenuto i risultati voluti in quanto non sono riuscito ad ottenere un modulo compatibile con il server avente estensione .dll. Di conseguenza non sono riuscito a testare il funzionamento del robot in ambiente Microsoft Windows.

Gli sviluppi futuri di questo elaborato potrebbero essere la compilazione dell'oggetto embedded, da collegare al server per interfacciare il Roboreptile, in ambiente Microsoft Windows e l'utilizzo della piattaforma URBI con il robot Lego Mindstorms NXT per applicazioni avanzate come ad esempio lo sviluppo di un Segway mantenuto in equilibrio trammite il simulatore di MatLab.

Bibliografia

[1] Kernel Team (2009), *The Urbi Software Development Kit*, Gostai, http://www.gostai.com/downloads/urbi-sdk-2.0/doc/urbi-sdk.pdf

[2] Benjamin Renoust, Benoit Pothier (2006), *URBI doc for LEGO Mindstorms NXT*, Gostai, http://www.gostai.com/doc/en/mindstormNXT.pdf

[3] Jean-Christophe Baillie (2007), *URBI Language Specification*, Gostai, http://www.gostai.com/doc/URBI-Specif-1.5.pdf

[4] *Software Developer Kit (SDK)*, Lego, http://mindstorms.lego.com/Overview/nxtreme.aspx

[5] *Bluetooth Developer Kit (BDK)*, Lego, http://mindstorms.lego.com/Overview/nxtreme.aspx

[6] Michael Collins (2006), *LEGO NXT Direct Commands API*, Collins, http://cpansearch.perl.org/src/COLLINS/NXT-1.40/lib/LEGO/NXT.pm

[7] *WowWee Roboreptile Technical Specifications*, WowWee, http://www.wowwee.com/en/products/toys/robots/robotics/robocreatures:roboreptile

[8] Evosapien (2006), *Roboreptile Review*, Evosapien, http://evosapien.com/robosapien-hack/nocturnal/RoboReptile/reveiw.html

[9] Pavel Petrovic, Richard Balogh (2006), *Controlling RoboSapien using LEGO IR-Tower*, Pavel, http://www.robotika.sk/mains.php?page=/projects/robsapien/

[10] *Wikipedia:l'Encilopedia Libera*, Wikimedia, http://it.wikipedia.org/wiki/Pagina_principale

[11] Walid Stefano Shajrawi (2008), Tesi, *Caratterizzazione dei sensori del robot Lego Mindstorms NXT*