Corso di Laurea Triennale in Ingegneria dell'Informazione

Analisi e commenti riguardanti programmi in LeJOS

Candidato:

Nicolò Siviero Ballini, 542447

Relatore:

Prof. Michele Moro

A.A. 2009/10

Indice

Capitolo 1

Introduzione

La robotica è una scienza che, partendo dallo studio dei comportamenti degli esseri intelligenti e mediante modellizzazione e semplificazione degli stessi, si occupa di dare ad una macchina (robot) delle capacità che possono essere, ad esempio, quelle di eseguire correttamente specifici compiti o di interagire correttamente (tramite sensori ed attuatori) con l'ambiente circostante, reagendo ad eventuali stimoli esterni. Già da questa semplice descrizione si ha l'idea della natura multidisciplinare che caratterizza tale scienza e, infatti, in essa confluiscono gli studi di molte discipline sia di natura umanistica come biologia, fisiologia, linguistica e psicologia che scientifica quali automazione, elettronica, fisica, informatica, matematica e meccanica.

Il progredire della tecnologia e l'abbassamento dei costi hanno favorito, nel tempo, l'ingresso in commercio di robot a un prezzo relativamente accessibile e in grado di offrire delle prestazioni soddisfacenti sia per quanto riguarda la capacità di calcolo che per la precisione dei sensori, e che hanno suscitato un grande interesse in ambito scolastico e accademico, nonché presso semplici appassionati, dovuto proprio al carattere interdisciplinare spiegato sopra e alla numerosità di problemi differenti che si possono affrontare nel modellare un comportamento e tentare di riprodurlo nel robot.

In questo contesto trovano da subito grande successo i robot della serie Mindstorms dell'azienda danese LEGO, caratterizzati da un'elevatissima flessibilità di costruzione, dalla buona capacità di calcolo dell'unità di elaborazione interna, dalla discreta precisione dei sensori e degli attuatori e infine da un ottimo rapporto potenzialità-flessibilità-prezzo. Dato il loro impiego sia presso scuole elementari che in luoghi accademici, sono stati sviluppati molteplici firmware che permettono ai robot LEGO di venire programmati tramite linguaggi più o meno ad alto livello, a seconda delle conoscenze e necessità dell'utente. Tra questi, si distingue LeJOS, una macchina virtuale Java semplificata e privata di alcune funzioni, che permette di gestire il brick LEGO e i suoi sensori e motori in linguaggio Java.

Lo scopo della presente tesina è quello di fornire degli esempi pratici e di varia natura, trovati mediante numerose ricerche in Internet, sia a chi ha basi del linguaggio Java ma si avvicina per la prima volta alla programmazione del brick NXT, sia a chi desidera avere algoritmi pronti o comunque idee risolutive che possano aiutare nel risolvere determinati problemi.

Nel capitolo 2 si presentano brevemente i brick LEGO, con particolare attenzione all'NXT, elencandone le caratteristiche tecniche e i sensori in dotazione, e inoltre si introduce sinteticamente il firmware sostitutivo LeJOS. Nel capitolo 3 si illustrano con semplici esempi l'uso dei sensori e dei motori che permettono al robot di interagire con l'ambiente e le varie tecniche di comunicazione Bluetooth con altri brick o con il PC. Nel capitolo 4 trovano posto alcuni esempi di robot inseguitori di linea e viene presentata l'importanza dell'uso dei behavior, dapprima in un semplice problema e successivamente all'interno di un progetto articolato in modo da rendere chiari i vantaggi derivanti dalla programmazione basata sulla Subsumption Architecture.

Capitolo 2

Breve descrizione dei robot LEGO Mindstorms e introduzione a LeJOS

2.1 Lego Mindstorms

Lego Mindstorms è una linea di prodotti Lego che consente di costruire robot e sistemi automatici interattivi più o meno complessi, grazie alla combinazione di mattoni programmabili, sensori di vario tipo, mattoni tradizionali, ingranaggi e parti pneumatiche.

La piena libertà di costruzione, garantita dalla grande varietà di pezzi per costruzioni LEGO che è possibile reperire in commercio, unita ad un costo non eccessivo dei kit Mindstorm, garantiscono sin dal 1998, anno di introduzione del brick programmabile RCX , un crescente successo di pubblico.

2.2 Il brick RCX

Il cuore della prima generazione di LEGO Mindstorms è stato il Robot Command Explorer (RCX), un mattone programmabile equipaggiato al suo interno da un microcontrollore Hitachi H8/3292, caratterizzato da 8 registri ad uso generale da 16 bit con spazio di indirizzamento anch'esso a 16 bit (dunque un totale di 64 kbytes per dati e programmi) e fornito di una memoria on-chip ROM + RAM di 16 Kbytes + 512 bytes. E' inoltre presente una memoria esterna di 32 Kbytes per il firmware e i programmi utente. Esternamente è dotato di un display LCD a 5 caratteri, 3 porte sensore analogiche a 10 bit e 3 porte motore, oltre ad una porta di comunicazione a infrarossi (IR) tramite la quale è possibile inviare i programmi al brick.

2.3 Il brick NXT

Introdotto nell'Agosto 2006, il brick NXT rappresenta la seconda e attuale generazione di Lego Mindstorms: esso è un'evoluzione dell'RCX, con caratteristiche hardware più avanzate e corredato da nuovi sensori; inoltre è stata tolta l'interfaccia a infrarossi in luogo delle più comode e comuni connessioni USB e Bluetooth.

2.3.1 Hardware

Le caratteristiche tecniche sono le seguenti:

- Microprocessore RISC a 32 bit Atmel AT91SAM7S256 a 48 MHz con 64 KB RAM e 256 KB flash memory, basato sull'architettura ARMv4T e studiato per dispositivi mobili e a bassa potenza;
- CoProcessore RISC a 8-bit ATmega48 a 4 MHz (4 KB flash memory e 512 Bytes RAM) dotato di un convertitore analogico/digitale a 10bit che agevola il controllo delle porte di input/output;

- Interfaccia bluetooth v2.0+EDR (chipset CSR BlueCore 4 version 2, con memoria RAM da 47 KBytes e firmware stack Bluelab 3.2) di velocità teorica massima 0,46 Mbit/sec, utile per trasferire il software o per controllare il robot da remoto e caratterizzata da un raggio di azione di circa 10 metri;
- Una porta USB 1.1 (velocità di 12 Mbit/s);
- Display LCD monocromatico con matrice 100x64 pixel;
- Speaker mono 8 bit;
- 4 porte di input a 6 fili di tipo simile a RJ12, che supportano la comunicazione tramite protocollo I^2C;
- 3 porte di output a 6 fili di tipo simile a RJ12.

L'alimentazione è garantita da sei batterie di tipo AA da 1,5 V ciascuna oppure da una batteria ricaricabile al litio, a seconda della versione del kit Mindstorm acquistata.

Il microprocessore Atmel, oltre ai compiti di calcolo aritmetico, gestisce direttamente le porte d'ingresso, la porta USB, l'altoparlante integrato ed è connesso al modulo Bluetooth via USART e SPI, tramite il quale controlla anche il display. La comunicazione con il microcontrollore ATmega48, che gestisce i servomotori, avviene per mezzo del bus I^2C; i diversi compiti svolti dalle due unità permettono di implementare via software una sorta di multitasking, cosicché mentre vengono ad esempio mossi i motori, l'AT91SAM7S256 può eseguire altre istruzioni, quali comunicazione con dispositivi periferici.

2.3.2 Sensori e servomotori

I principali sensori prodotti dalla LEGO, che vanno collegati alle porte di input del brick NXT, sono i seguenti:

1. **Sensore di contatto**
   Di tipo passivo, fornisce un segnale di tipo ON-OFF (0 se rilasciato, 1 se premuto). È possibile impostare il sensore per leggere il valore grezzo da 0 a 1023.
2. **Sensore di luce**
   Il segnale fornito è di tipo analogico ed è proporzionale alla luminosità registrata in una direzione. E' possibile riconoscere varie tonalità di grigio e rilevare diverse intensità di luce ambiente o di luce riflessa nel caso si accenda il led incorporato nel sensore.
3. **Sensore sonoro**
   Fornisce un segnale di tipo analogico proporzionale all'intensità della pressione sonora rilevata; può misurare livelli di suono sino a circa 90 dB. Essendo molto complicato rilevare un valore assoluto, questo viene indicato in percentuale: ad esempio segnala circa 4-5% per un rumore di fondo, 10-30% per una normale conversazione vicina al microfono sino a 40-100% per urla o musica ad alto volume.
4. **Sensore a ultrasuoni**
   Rileva la distanza da un ostacolo misurando il tempo impiegato da un'onda sonora a colpirlo e tornare indietro. E' in grado di valutare distanze da 0 a 235 cm, con un'approssimazione di circa ±3 cm. Ovviamente la rilevazione gode di maggior precisione con oggetti grandi e aventi superfici dure, rispetto a ostacoli piccoli e di forma irregolare.
   Da notare che l'uso contemporaneo e vicino di due sensori di questo tipo può rendere impreciso il loro funzionamento.

Esistono inoltre in commercio sensori meno comuni quali l'accelerometro, un rilevatore di infrarossi e di comunicazioni a infrarossi e una bussola.

I sensori passivi vengono campionati ogni 3 ms, mentre a quelli attivi viene fornita potenza per 3 ms e si utilizzano gli 0,1 ms immediatamente successivi per rilevare il dato.

Un altro componente di fondamentale importanza per l'NXT è rappresentato dal servomotore, funzionante in DC, che include al suo interno un sensore di rotazione di sensibilità pari a un grado tramite il quale si possono rilevare indirettamente velocità e distanza percorsa dal robot, oltre a rendere maggiormente preciso il controllo del motore stesso. Le caratteristiche tecniche, riferite a una tensione di alimentazione di 9V, sono le seguenti:

- Velocità 117 rpm (max 170 rpm);
- Potenza meccanica ed elettrica rispettivamente di 2,03W e 4,95W;
- Efficenza del 41%;
- Assorbimento 0,55°;
- No-Load current 60mA;
- Coppia di 16,7N*cm;
- Coppia e corrente di stallo rispettivamente di 50N*cm e 2°;
- Peso 80 gr.

2.4 Storia ed evoluzione di LeJOS

LeJOS (Lego Java Operating System) è un firmware sostitutivo per i brick programmabili LEGO RCX e NXT, basato su un progetto open source di José Solórzano il cui scopo iniziale era quello di creare una TinyVM, ovvero una Java Virtual Machine opportunamente semplificata e adattata in modo da poter funzionare egregiamente anche con limitate risorse e in particolare con i sensori e l'hardware del brick. In sostanza LeJOS permette di eseguire codice Java all'interno dei brick, aumentandone di molto le potenzialità rispetto alla presenza del firmware originario.

Dopo l'uscita a fine Settembre 2006 dell'ultima release per l'RCX in concomitanza con la sostituzione in commercio del brick, inizia lo sviluppo di LeJOS rivolto al nuovo NXT e ne viene rilasciata all'inizio dell'anno successivo la prima versione.

LeJOS NXJ usa la stessa Java Virtual Machine della versione RCX ma consta di driver dedicati all'hardware del nuovo brick e aggiunge nuove funzionalità quali un menù di sistema, supporto ai sensori I^2C, nonché la possibilità di usare le comunicazioni via Bluetooth e USB e di sfruttare pienamente la maggior precisione dei nuovi servomotori permettendo all'utente finale un uso più accurato degli stessi.

2.5 Principali caratteristiche

Rispetto al firmware originale, le principali caratteristiche che LeJOS mette a disposizione dell'utente sono:

- Array, anche multidimensionali;
- Preemptive threads (tasks);
- Ricorsione;
- Eccezioni e loro gestione;
- Alcuni tipi Java tra i quali float, long e string (che permettono di utilizzare con buona precisione le funzioni trigonometriche, particolarmente utili per gestire il movimento del robot) ma con alcune limitazioni;

- La maggior parte delle classi di java.lang, java.util e java.io.

Sono inoltre presenti delle API interamente documentate e corredate da esempi per facilitare l'utente nel loro utilizzo.

2.6 Limitazioni

Le limitate risorse hardware del brick NXT, come è lecito aspettarsi, pongono dei limiti alle potenzialità che Java esprime in presenza di hardware con maggiori prestazioni.

Alcune tra le principali restrizioni sono dovute alla memoria: in primo luogo si nota che LeJOS occupa circa 27 KB, che di conseguenza lascia a disposizione dell'utente 229 KB per i programmi e i dati. Anche la tecnica di programmazione della ricorsione viene penalizzata dalla scarsa memoria, che rende possibile solo dieci livelli di profondità, che possono eventualmente diminuire al crescere del numero di variabili locali utilizzate. Infine gli array, siano essi multidimensionali o no, possono avere una dimensione massima di 511 elementi.

Un'altra importante limitazione da tener presente in fase di programmazione, dovuta sia all'esigenza di non sovraccaricare di lavoro la CPU (che si traduce in maggior risparmio energetico e in maggiore velocità di esecuzione dei programmi) sia, di nuovo, alla scarsa memoria, è la mancanza del Garbage Collector, un processo che elimina gli oggetti creati da un programma quando questi non vengono più utilizzati. Se la mancanza di questo processo da un lato libera le risorse di sistema da una buona quantità di lavoro, dall'altra obbliga l'utente a prestare attenzione all'uso della parola chiave *new*, ad esempio all'interno di cicli dei quali non si può conoscere in anticipo il numero di iterazioni, poiché troppe creazioni di oggetti porterebbero al riempimento della memoria e al conseguente errore in esecuzione.

Capitolo 3

Interazione del brick NXT con l'ambiente esterno: esempi di uso di sensori, motori e Bluetooth

3.1 Esempi di uso dei sensori con LeJOS

3.1.1 Sensore di contatto

Il sensore di contatto è quello, tra i quattro presentati, con il funzionamento più semplice e rappresenta quindi l'ideale per una prima piccola analisi.
Posizionamento sensori e motori: sensore di contatto collegato alla porta 1.
Scopi ed effetti del programma: viene visualizzata sul display la stringa "Finished" quando viene premuto il sensore di contatto.

```
import lejos.nxt.* ;

public class TouchTest {
        public static void main ( String [ ] args ) throws Exception {
                TouchSensor touch = new TouchSensor( SensorPort.S1 ) ;
                while ( !touch.isPressed( ) ) ;
                        LCD.drawString( "Finished", 3, 4) ;
                }
        }
}
```

Commenti e note:

- Qualunque sia il tipo di sensore, è necessario indicare sempre nel costruttore la porta a cui è collegato.
- I parametri numerici di LCD.drawString sono le coordinate (rispettivamente ascissa e ordinata con punto d'origine posizionato in alto a sinistra del display) alle quali si vedrà apparire il testo; l'unità di misura delle coordinate sono i caratteri (0÷15 caratteri in larghezza, 0÷7 in altezza).

3.1.2 Sensore di luce

In questo esempio vengono presentati vari modi per leggere quanto rilevato dal sensore di luce.
Posizionamento sensori e motori: sensore di luce collegato alla porta 1.
Scopi ed effetti del programma: sul display vengono continuamente visualizzati i valori di letture effettuate con metodi diversi.

```
import lejos.nxt.* ;

public class LightTest {
        public static void main( String [ ] args ) throws Exception {
                LightSensor light = new LightSensor( SensorPort.S1) ;
                while ( true ) {
                        LCD.drawInt( light.getLightValue( ), 4, 0, 0) ;
                        LCD.drawInt( light.getNormalizedLightValue( ), 4, 0, 1) ;
                        LCD.drawInt( SensorPort.S1.readRawValue( ), 4, 0, 2) ;
                        LCD.drawInt( SensorPort.S1.readValue( ), 4, 0, 3) ;
                }
        }
}
```

Commenti e note:

- Il costruttore della classe LightSensor presenta una particolarità: così come è utilizzato nel programma sottintende che il sensore verrà utilizzato con il led incorporato acceso. Se si intende fare diversamente, ovvero un uso con led spento, è necessario utilizzare il secondo costruttore a disposizione, ovvero LightSensor(ADSensorPort port, boolean floodlight), con la variabile booleana pari a *false*.
- Il metodo getLightValue( ) restituisce la percentuale di luce rilevata, da 0 (buio) a 100 (luce diretta del sole); getNormalizedLightValue( ), invece, fornisce il valore della luminosità in una scala da 0 a 1023: tipicamente il sensore LEGO invia valori compresi tra 145 e 890.
- I metodi readValue( ) e readRawValue( ) eseguono rispettivamente gli stessi compiti dei metodi descritti al punto precedente; vengono però sfruttati i metodi definiti dall'interfaccia ADSensorPort che SensorPort implementa. La peculiarità consiste nel fatto che i due metodi si possono utilizzare per leggere i valori delle porte (la 1 in questo specifico caso) qualsiasi siano i sensori collegati.
- I parametri di LCD.drawInt sono rispettivamente un intero, il numero di caratteri da visualizzare e infine le coordinate x e y del display.
- Questo programma d'esempio può essere utilizzato per capire la sensibilità e il comportamento del sensore in varie condizioni di luce per permettere successivamente all'applicazione che ne fa uso un più accurato controllo.

3.1.3 Sensore sonoro

Posizionamento sensori e motori: sensore acustico collegato alla porta 2.
Scopi ed effetti del programma: sul display viene visualizzata, a intervalli di 300 ms, la percentuale di livello sonoro rilevata.

```
import lejos.nxt.*;

public class SoundSensorTest{
        public static void main ( String [ ] args ) throws Exception {
                SoundSensor ss = new SoundSensor( SensorPort.S2, false) ;
                LCD.drawString( "Sound level (%) ", 0, 0) ;
                while ( !Button.ESCAPE.isPressed( ) ) {
                        LCD.drawInt( ss.readValue( ), 3, 13, 0) ;
                        LCD.refresh( ) ;
                        Thread.sleep( 300) ;
                }
                LCD.clear( ) ;
                LCD.drawString( "Program stopped", 0, 0) ;
                LCD.refresh( ) ;
        }
}
```

Commenti e note:

- Il parametro booleano nel costruttore di SoundSensor indica se il sensore funziona o meno in dBA. Esistono infatti due modi per utilizzare il sensore: quando è impiegato come sopra, esprime una misura in dB (percentuale) e percepisce suoni anche molto deboli che l'orecchio umano non è in grado di sentire, mentre se lo si utilizza per la rilevazione in dBA(Adjusted Decibel) esso funziona nel medesimo range di suoni che l'apparato acustico umano può udire. In quest'ultimo caso, quindi, il sensore si comporta come una sorta di orecchio artificiale.

  Per capire a fondo il diverso funzionamento nelle due diverse modalità, si può passare da un modo all'altro utilizzando il metodo setDBA (boolean dBA).

3.1.4 Sensore a ultrasuoni

Posizionamento sensori e motori: sensore a ultrasuoni collegato alla porta 1.
Scopi ed effetti del programma: sul display viene visualizzata la distanza in cm dell'ostacolo considerato dal brick a intervalli di 300 ms; la pressione del tasto Esc dell'NXT termina l'esecuzione.

```
import lejos.nxt.* ;

public class SonicSensorTest {
        public static void main (String [ ] args) throws Exception {
                UltrasonicSensor us = new UltrasonicSensor( SensorPort.S1) ;
                LCD.drawString( "Distance(cm) ", 0, 0) ;
                while (! Button.ESCAPE.isPressed()) {
                        LCD.drawInt(us.getDistance(),3,13,0);
                        Thread.sleep( 300) ;
                }
                LCD.clear( ) ;
                LCD.drawString( "Program stopped", 0, 0) ;
                Thread.sleep( 2000) ;
        }
 }
```

Commenti e note:

- Ci sono due differenti modalità di utilizzo del sensore di distanza: continua e ping. Nell'esempio viene presentata la prima, nella quale il sensore invia un'onda sonora e appena viene rilevato il ritorno, ne viene spedita una nuova e così via di continuo.
- L'istruzione Thread.sleep(300), che ferma il programma per i corrispondenti millisecondi, è necessaria poiché è una limitazione relativa alla release 0.3 di LeJOS, dovuta sia a problemi software sia in parte ad un piccolo contributo di tempo necessario all'onda sonora per compiere il tragitto brick-ostacolo-brick.
- Sperimentalmente si è osservato che la distanza massima rilevata dal sensore è tra i 170 e i 210 cm; il valore di 255 cm viene utilizzato per indicare che non sono stati percepiti ostacoli nel raggio di funzionamento.

Il modo di funzionamento in ping è migliore sia, anche se in minima parte, sotto l'aspetto dell'autonomia energetica del robot, sia per quanto riguarda la minimizzazione dei conflitti presenti tra più sensori di distanza che funzionino contemporaneamente.

Posizionamento sensori e motori: sensore a ultrasuoni collegato alla porta 3.
Scopi ed effetti del programma: sul display vengono visualizzati progressivamente su ogni colonna gli otto valori letti in modalità ping. Premendo il tasto Escape il programma termina.

```
import lejos.nxt.* ;
public class USPingTest{
        public static void main( String[ ] args) {
                LCD.drawString( " USPing", 0, 0) ;
                Button.waitForPress( ) ;
                LCD.clear( ) ;
                UltrasonicSensor sonar = new UltrasonicSensor( SensorPort.S3) ;
                int[ ] distances = new int[8] ;
                int col = 0 ;
                boolean more = true ;
                while ( more) {
                        sonar.ping( ) ;
                        Sound.pause( 30) ;
                        sonar.getDistances( distances) ;
                        for ( int i = 0; i<distances.length ;i++) {
                                LCD.drawInt( distances[i], 4, 4*col, i) ;
                        }
                        col++ ;
                        if ( col >4) {
                                more = Button.waitForPress( )<8 ;
                                col = 0 ;
                        }
                }
        }
}
```

Commenti e note:

- In questa modalità di utilizzo il sensore invia una sola onda e salva sino a 8 tempi di risposta (convertite in distanze). Quindi se vengono ad esempio posti due oggetti, uno a sinistra e uno a destra rispetto alla linea centrale di rilevazione, si dovrebbero ottenere nei primi due valori la distanza di tali ostacoli.
- L'invio dell'onda avviene solo in corrispondenza del metodo ping( ).
- Poiché è obbligatorio aspettare approssimativamente almeno 20 ms tra la chiamata a ping( ) e getDistances(int [ ] dist) è stato inserita l'istruzione Sound.Pause(30) che lascia intercorrere 30 ms tra le due istruzioni che separa.
- Si può utilizzare anche il metodo getDistance( ), in questo caso però verrà restituito soltanto il primo valore delle otto distanze (quella più piccola).

3.2 Esempi di uso dei servomotori

3.2.1 Robot scaccia intrusi

Posizionamento sensori e motori: un sensore a ultrasuoni collegato alla porta 2 e due servomotori connessi alle porte B e C. Si consiglia di costruire una chela elementare (dal lato delle porte numerate, ovvero dove è installato il sensore) per riuscire a spostare gli oggetti senza che essi scivolino via.
Scopi ed effetti del programma: il robot gira su se stesso analizzando lo spazio attorno a sé tramite il sensore preposto alla ricerca di ostacoli in un raggio pari a quanto indicato dalla costante guardDistance. Individuato un eventuale ostacolo, il robot si muove verso di esso, lo porta fuori dal proprio raggio d'azione e ritorna alla propria posizione iniziale dalla quale, una volta arrivato, continua a scansionare lo spazio circostante. Il programma termina quando viene premuto un qualsiasi tasto del brick.

```
import lejos.nxt.*;
public class MotorTest {
	public static void main(String [] args) {
		final int pushTime = 3000;
		final int guardDistance = 75;
		// wait three seconds
		for (int i=0;i<6;i++) {
			Sound.beep();
			try { Thread.sleep(500); } catch (InterruptedException e) {}
		}
		UltrasonicSensor uss = new UltrasonicSensor(SensorPort.S2);
		LCD lcd = new LCD();
		while (Button.readButtons() == 0) {
			// rotate in opposite ways
			Motor.C.forward();
			Motor.B.backward();
			// set speeds
			Motor.B.setSpeed(200);
			Motor.C.setSpeed(200);
			// ping
			uss.ping();
			try { Thread.sleep(20); } catch (InterruptedException e) {}
			// check for an object
			int dist = uss.getDistance();
			lcd.clear();
			lcd.drawInt(dist, 0, 0);
			if (dist <= guardDistance) {
				// attack!
				Motor.B.backward();
				Motor.C.backward();
```

```
                        // set speeds
                        Motor.B.setSpeed(800);
                        Motor.C.setSpeed(800);
                        // wait for a little while
                        try { Thread.sleep(pushTime); } catch (InterruptedException e) {}
                        Motor.B.forward();
                        Motor.C.forward();
                        // wait for a little while
                        try { Thread.sleep(pushTime); } catch (InterruptedException e) {}
                    } //end if
                    // wait for a little while
                    try { Thread.sleep(100); } catch (InterruptedException e) {}
                } //end while
                // stop the motor
                Motor.C.stop();
                Motor.B.stop();
        }
}
```

Commenti e note:

- Il movimento del robot atto a spostare l'ostacolo fuori dal proprio raggio d'azione e a ritornare alla posizione iniziale è ottenuto muovendo i motori indietro e avanti per lo stesso intervallo di tempo: si potrebbe pensare che non è così scontato il fatto che il brick torni al punto iniziale dato che durante il tragitto di andata deve spostare un certo peso che lo potrebbe rallentare, ma grazie a setSpeed(int speed) ciò è effettivamente garantito poiché, anche se viene incontrato un ostacolo di massa non trascurabile, setSpeed mantiene costante la velocità (indicata in gradi per secondo) dei motori, batteria permettendo.

  In realtà un certo grado di imprecisione è dato dall'impatto dell'oggetto contro il robot che può leggermente modificarne la traiettoria: un possibile rimedio è quello di aumentare la massa del robot e di diminuire la velocità in modo che risenta in tono minore di questo problema.
- La velocità massima che il brick può sostenere è di circa 900 gradi per secondo con batterie cariche (110 volte la tensione di alimentazione).

La rotazione su se stesso del robot è ottenuta semplicemente impostando la stessa velocità su entrambi i motori e facendoli girare uno in verso opposto all'altro. Come si potrà notare nei prossimi esempi, ci sono vari modi per realizzare ciò e in generale sono presenti altri metodi per far ruotare i servomotori che sfruttano il sensore di rotazione interno ad essi (quest'ultimo, si ricorda, non era presente nei motori di prima generazione dedicati al brick RCX).

3.2.2 Un radar primitivo

Posizionamento sensori e motori: sensore a ultrasuoni collegato alla porta 1, servomotori collegati alle porte A e B.

Scopi ed effetti del programma: sul display viene visualizzata l'analisi dello spazio circostante: ciascuna colonna di pixel sul display rappresenta uno spostamento su se stesso del robot ottenuto dall'avanzamento di due gradi dei servomotori mentre la quantità di pixel neri su ciascuna colonna è direttamente proporzionale alla distanza del brick da eventuali ostacoli (quindi una colonna bianca indica che è stato rilevato un ostacolo molto vicino). Dopo aver effettuato un intero giro su se stessi, i servomotori riportano il robot alla posizione di partenza. Premendo il tasto Escape il programma termina.

```
import lejos.nxt.*;

public class RadarSimple {
        public static void main(String [] args) throws Exception{
                UltrasonicSensor u1 = new UltrasonicSensor(SensorPort.S1);
                int dist, x = 0 ,y = 0;
                u1.continuous();
                for(int i = 0; i <= 180; i++){
                        dist = u1.getDistance();
                        x= (i*LCD.SCREEN_WIDTH)/180;
                        y= (dist*LCD.SCREEN_HEIGHT)/255;
                        for(int j = y; j <= LCD.SCREEN_HEIGHT; j++){
                                LCD.setPixel(255, x, j);
                        }
                        Motor.A.rotateTo(-2*i,true);
                        Motor.B.rotateTo(2*i);
                }
                Motor.A.rotateTo(0,true);
                Motor.B.rotateTo(0);
                Button.ESCAPE.waitForPressAndRelease();
                Thread.sleep(1000);
        }
}
```

Commenti e note:

- Il parametro del metodo rotateTo(int limitAngle) non è relativo, ovvero all'inizio del programma viene assunto che i motori siano entrambi posizionati al grado 0 e un'invocazione di rotateTo ha l'effetto di portare il motore all'angolo specificato in relazione proprio all'angolo di partenza 0. Se, ad esempio, vengono consecutivamente eseguite due istruzioni identiche quali rotateTo(360) si vedrà il servomotore compiere un intero giro all'esecuzione della prima, e stare fermo all'esecuzione della seconda. Infatti, alla fine del programma usando rotateTo(0) per entrambi i motori essi riportano il robot alla posizione iniziale.
- Se si vuole ruotare il robot di un determinato angolo, è disponibile il metodo rotate (int angle) che ruota i motori di volta in volta di quanto specificato nel parametro d'ingresso. A titolo d'esempio,

se in questo caso alla fine del programma ci fossero state due invocazioni di rotate(0) in luogo di rotateTo(0), il robot sarebbe rimasto fermo.

- Il metodo rotateTo(int limitAngle, boolean immediateReturn) con il parametro booleano pari a *true* permette che la rotazione sino a limitAngle possa essere interrotta da altri comandi che fanno uso del servomotore in questione. Sostanzialmente la rotazione viene avviata e, se nelle istruzioni successive ci sono altri comandi che fanno uso dello stesso servomotore, essi vengono eseguiti senza attendere che venga portata a termine.
- Nell'immagine viene mostrato il display del brick dopo aver eseguito il programma: le colonne bianche indicano la presenza di un ostacolo molto ravvicinato nello spicchio centrale di rilevamento.

Figura 1 : display dell'NXT dopo aver eseguito il programma RadarSimple.

3.2.3 Robot evita ostacoli

Posizionamento sensori e motori: sensore a ultrasuoni collegato alla porta 1, servomotori collegati alle porte A e B.

Scopi ed effetti del programma: sul display viene visualizzato un conto alla rovescia, successivamente il robot avanza sino a che non incontra un ostacolo ad una distanza minore di 25 cm. Quando ciò accade esso decide casualmente se sterzare verso destra o sinistra per evitarlo e riprende ad avanzare e così via.

```
import java.util.Random;
import lejos.nxt.LCD;
import lejos.nxt.Motor;
import lejos.nxt.SensorPort;
import lejos.nxt.UltrasonicSensor;

public class WallAvoider {
	final static UltrasonicSensor ultrasonic = new UltrasonicSensor(SensorPort.S1);
	public static void main(String args[]) throws Exception {
		LCD.drawString("3",3,4);
		Thread.sleep(1000);
		LCD.clearDisplay();
		LCD.drawString("2",3,4);
		Thread.sleep(1000);
		LCD.clearDisplay();
		LCD.drawString("1",3,4);
		Thread.sleep(1000);
		LCD.clearDisplay();
		Motor.A.setSpeed(750);
		Motor.B.setSpeed(750);
		while(true) {
			LCD.clearDisplay();
			LCD.drawInt(ultrasonic.getDistance(), 3, 3);
			Motor.A.forward();
			Motor.B.forward();
			distanceTest();
		}
	}
	public static void distanceTest() {
		LCD.clearDisplay();
		LCD.drawInt(ultrasonic.getDistance(), 3, 3);
		while(true) {
			if(ultrasonic.getDistance() < 25) {
				if(right()) {
					Motor.A.stop();
					Motor.B.stop();
					Motor.A.rotate(180,true);
					Motor.B.rotate(-180);
					Motor.A.stop();
					Motor.B.stop();
				}else {
					Motor.A.stop();
					Motor.B.stop();
					Motor.A.rotate(-180,true);
```

```
                                        Motor.B.rotate(180);
                                        Motor.A.stop();
                                        Motor.B.stop();
                                    } //endif
                                    break;
                                }//endif
                        }//endwhile
                }
                private static boolean right() {
                        Random diceRoller = new Random();
                        int roll = diceRoller.nextInt(2) + 1;
                        if (roll == 1) return true;
                        else return false;
}
}
```

Commenti e note:

- La classe WallAvoider non è adatta ad essere usata, ad esempio, in una stanza con molti ostacoli poiché se il robot ad ogni svolta si trova immediatamente un altro ostacolo può accadere che, a causa della scelta casuale, esso si giri nuovamente nella direzione di quello precedente.

3.3 L'interfaccia Pilot

Dagli esempi precedenti inerenti i servomotori si evince che la precisione con cui vengono eseguiti i comandi riguardanti questi ultimi non si traduce poi in una altrettanta e immediata precisione quando si tratta di far girare il robot di un determinato angolo, oppure di fargli eseguire un determinato percorso. Infatti per passare da uno spostamento angolare del motore a uno spostamento o rotazione effettiva del robot sulla superficie sulla quale è posto entrano in gioco diversi fattori, quali ad esempio il diametro delle ruote ad essi applicate o la distanza tra di esse, cioè parametri che variano a seconda della particolare forma che assume di volta in volta il robot e che richiederebbero, ad esempio in un programma il cui scopo è quello di disegnare una circonferenza, di modificare di volta in volta in più punti, e per giunta in modo abbastanza sperimentale, il programma a seconda della particolare costruzione del robot sul quale esso viene memorizzato.

Se in molti casi, ad esempio nel programma del sottocapitolo 3.2.1, non è necessaria una precisione come quella di cui si è discusso precedentemente, è tuttavia lampante come in altrettanti questa si renda necessaria nel caso si voglia anche solo far compiere dei semplici percorsi geometrici a un robot; per rispondere a questa necessità LeJOS mette a disposizione l'interfaccia Pilot: nel costruttore delle tre classi che la implementano vengono date le informazioni riguardanti la posizione dei motori, il diametro delle ruote e la distanza tra di esse, in modo tale da rendere i movimenti del robot estremamente precisi (l'errore massimo si aggira intorno al 2%) e di semplificare notevolmente la portabilità di un programma a robot costruiti diversamente da quello originale. E' infatti sufficiente modificare solamente i parametri passati al costruttore indicando le caratteristiche costruttive del nuovo robot per ottenere un'analoga precisione di movimento.

Seguono alcuni esempi di utilizzo della classe Pilot.

3.3.1 Uso di TachoPilot: disegnare forme geometriche

Posizionamento sensori e motori: servomotori alla porta A e C.
Scopi ed effetti del programma: Il robot percorre un quadrato, una circonferenza inscritta all'interno di esso e infine un otto (quest'ultimo dopo essersi portato al centro delle figure precedenti).

```
import lejos.nxt.*;
import lejos.robotics.navigation.*;

public class Figure {
	public static void main(String[] args) throws Exception {
		Pilot navigator = new TachoPilot(5.6f, 11.4f, Motor.A, Motor.C);
		navigator.reset();
		//quadrato
		navigator.travel(30);
		navigator.rotate(90);
		navigator.travel(30);
		navigator.rotate(90);
		navigator.travel(30);
		navigator.rotate(90);
		navigator.travel(30);
		navigator.rotate(90);

		//ricollocazione
		Thread.sleep(500);
		navigator.travel(15);
		Thread.sleep(500);

		//circonferenza
		navigator.arc(15,360);

		//ricollocazione
		Thread.sleep(500);
		navigator.rotate(90);
		navigator.travel(15);
		Thread.sleep(500);

		//8
		navigator.arc((float) 6.25,360);
		navigator.arc((float) -6.25,360);

		LCD.drawInt( (int) navigator.getTravelDistance(), 6,4);
		Button.waitForPress();
	}
}
```

Commenti e note:

- Il costruttore TachoPilot(float wheelDiameter, float trackWidth, TachoMotor leftMotor, TachoMotor rightMotor) richiede in ingresso il diametro e la distanza tra le due ruote, oltre alle porte dove sono collegati fisicamente i motori. Le misure richieste possono essere date in qualsiasi unità di misura, a patto che sia la stessa per entrambi i parametri. Un altro costruttore prevede anche la possibilità che le due ruote abbiano diametri differenti tra loro.
- Se il parametro angle dei metodi rotate(float angle) e arc(float radius, float angle) è positivo il robot gira verso sinistra (senso antiorario), se negativo verso destra (orario).
- Nei metodi tutti i parametri che indicano una distanza possono essere espressi in qualsiasi unità di misura purché sia la medesima utilizzata nel costruttore.

3.3.2 Navigator: arrivo a destinazione evitando eventuali ostacoli

La classe SimpleNavigator, il cui costruttore richiede come parametri una classe che implementi Pilot oppure, come in TachoMotor, direttamente i parametri costruttivi del robot(quest'ultima soluzione è deprecata dato che, essendo la classe in oggetto una astrazione di un semplice navigatore, dovrebbe essere indipendente da qualsiasi parametro costruttivo caratterizzante il veicolo che guida), permette di portare un robot ad una qualsiasi coordinata (x,y) del piano su cui esso si muove. La posizione e la direzione di partenza vengono riconosciute rispettivamente come l'origine e l'orientamento dell'asse delle ascisse e quando il brick si muove le coordinate vengono calcolate facendo uso dei sensori interni ai motori in combinazione alle caratteristiche costruttive del robot. Questa soluzione, se da una parte non prevede uso di sensori aggiuntivi, permette una completa libertà di costruzione e comporta pochi calcoli da parte dell'hardware, dall'altra comporta svantaggi nella stima della posizione corrente che può progressivamente diventare imprecisa all'aumentare della distanza percorsa dal robot.
Posizionamento sensori e motori: servomotori alle tre porte A, B e C. Il motore B serve a far ruotare verso sinistra o destra il sensore a ultrasuoni collegato alla porta 3.
Scopi ed effetti del programma: collocato il robot tenendo presente che il punto di partenza rappresenta l'origine e la direzione in cui è posto il verso dell'asse delle ascisse, esso raggiunge le coordinate di arrivo specificate aggirando gli eventuali ostacoli incontrati lungo il percorso grazie ad un algoritmo in parte casuale.

```
import lejos.nxt.*;
import lejos.robotics.navigation.*;
import java.util.Random;

public class EchoNavigator{
        public EchoNavigator(SimpleNavigator navigator, SensorPort echo, Motor scanMotor){
                this.navigator = navigator;
                sonar= new UltrasonicSensor(echo);
                scanner = scanMotor ;
        }
        /* attempt to reach a destinaton at coordinates x,y despite obstacle.
        @param x coordinate of destination
        @param y coordinate of destination. */
```

```
	public void goTo(float x, float y){
		navigator.setMoveSpeed(20);
		navigator.setTurnSpeed(180);
		float destX = x;
		float destY = y;
		while (navigator.distanceTo(destX,destY) > 5){
			navigator.goTo(destX, destY,true);
			boolean clear  = readDistance();
			if (!clear){ //obstacle found
				while (!avoid()) Thread.yield();  // keeps calling avoid until no
obstacle is in view
			}
		}
	}
	/* backs up, rotates away from the obstacle, and travels forward; returns true if no obstacle
	was discovered while traveling */
	private boolean avoid(){
		int leftDist = 0;
		int rightDist = 0;
		byte turnDirection = 1;
		boolean more = true;
		while(more){
			scanner.rotateTo(75);
			Sound.pause(20);
			leftDist = sonar.getDistance();
			scanner.rotateTo(-70);
			Sound.pause(20);
			rightDist = sonar.getDistance();
			if(leftDist>rightDist) turnDirection = 1;
			else turnDirection = -1;
			more = leftDist < _limit && rightDist< _limit;
			if(more) navigator.travel(-4);
			LCD.drawInt(leftDist,4,0,5);
			LCD.drawInt(rightDist,4,8,5);
		}
		scanner.rotateTo(0);
		navigator.travel(-10 - rand.nextInt(10));
		int angle = 60+rand.nextInt(60);
		navigator.rotate(turnDirection * angle);
		navigator.travel(10 + rand.nextInt(60), true);
		return  readDistance (); // watch for hit while moving forward
	}
	/* Monitors the ultrasonic sensor while the robot is moving.
	Returns if an obstacle is detected or if the robot stops;
```

```
        Returns false if obstacle was detected */
        public boolean readDistance(){
                System.out.println(" Moving ");
                int distance = 255;
                boolean clear = true;
                while( navigator.isMoving()& distance > _limit ){
                        distance = sonar.getDistance();
                        LCD.drawString("D "+distance, 0, 0);
                        clear = distance > _limit ;
                        Thread.yield();
                }
                navigator.stop();
                return clear;
        }
        public static void main(String[] args){
                System.out.println("Any Button");
                TachoPilot p = new TachoPilot(5.6f, 14.2f, Motor.A, Motor.C);
                EchoNavigator  robot  = new EchoNavigator( new SimpleNavigator(p),
SensorPort.S3, Motor.B);
                Button.waitForPress();
                robot.goTo(200,0);
        }

        public SimpleNavigator navigator ;
        Random rand = new Random();
        UltrasonicSensor sonar;
        private Motor scanner;
        int _limit =20; //cm
}
```

Commenti e note:

- In accordo con quanto detto poco sopra, la precisione con cui il robot raggiunge la destinazione impostata degrada all'aumentare degli ostacoli trovati sulla propria strada. E' necessario impostare con molta cura i parametri della classe TachoPilot, ovvero le caratteristiche costruttive.
- La classe SimpleNavigator può eseguire solamente tre movimenti elementari, ovvero viaggiare su una linea retta, muoversi su un arco di circonferenza e ruotare da fermo.
- Le coordinate da raggiungere, in analogia all'esempio precedente, devono essere espresse nella medesima unità di misura usata per indicare la distanza e il diametro delle ruote del robot.

3.4 Bluetooth

Il brick NXT, come già indicato in precedenza, permette la comunicazione sia via Bluetooth sia via USB. Indubbiamente quest'ultima è molto più veloce ma si può utilizzare soltanto per comunicazioni NXT-PC e viceversa, oltre ad avere un limite che nel campo della robotica può essere davvero molto difficile da accettare, ovvero quello di essere una connessione di tipo wired. D'altro canto il Bluetooth non fornisce la stessa velocità, ma permette una notevole flessibilità (sono possibili ad esempio comunicazioni NXT-NXT, PC-NXT, telefono cellulare-NXT) ed è una connessione di tipo wireless che permette, quindi, una maggiore libertà di movimento al brick.

LeJOS NXJ mette a disposizione, nei package lejos.pc.comm e lejos.nxt.comm, delle classi concepite e strutturate in modo che la maggior parte del codice sia indipendente dal fatto che si usi l'una o l'altra trasmissione.

Il primo passo per permettere una comunicazione tra due dispositivi è di creare una connessione: essa viene inizializzata da una delle due periferiche in gioco mentre l'altra si mette in ascolto (ricerca) della connessione creata. Quando viene stabilita, entrambe le estremità della connessione possono aprire un flusso (stream) in input o output per leggere o scrivere dati, come di consueto nel linguaggio Java. È importante che tutti i dispositivi che comunicano con l'NXT tramite Bluetooth supportino il profilo SPP (Bluetooth Serial Port Profile), l'unico che il brick è in grado di riconoscere.

Affinché la comunicazione Bluetooth funzioni, è necessario che ogni NXT o dispositivo abbia un nome unico, e in aggiunta che la periferica che inizialmente sarà alla ricerca della connessione sia conosciuta da quella che la inizializzerà. Per fare ciò, ad esempio tra due NXT, è necessario assicurarsi preventivamente che il Bluetooth del brick che in un primo momento si può identificare come ricevitore sia attivo di modo che, andando nel menù Bluetooth dell'NXT che inizializza la comunicazione e attivando la funzione cerca ("search"), il suo nome compaia tra i dispositivi trovati e lo si possa aggiungere ("add").

Questo è un passo obbligato ogni qualvolta si voglia utilizzare una comunicazione di tipo Bluetooth, ovviamente con le opportune modifiche a seconda del tipo di dispositivi coinvolti e del rispettivo ruolo all'inizio della connessione all'interno del programma LeJOS. E' preferibile che, in caso di comunicazione NXT-PC, sia quest'ultimo a inizializzare la connessione; invece è opportuno lasciare il ruolo di inizializzazione all'NXT solo, ovviamente, nel caso esso abbia a che fare con altri brick o con dispositivi che non sono in grado di creare una connessione ma soltanto di mettersi in ascolto di quest'ultima.

3.4.1 Semplice comunicazione NXT-NXT

Posizionamento sensori e motori: nessun motore o sensore collegato.
Scopi ed effetti dei programmi: le due classi che vengono illustrate di seguito costituiscono un punto di partenza e in particolare sono un esempio di una semplice comunicazione via Bluetooth tra due brick NXT: in questo caso vengono trasmessi degli interi in entrambe le direzioni e quanto ricevuto viene visualizzato sul display. E' ovviamente possibile, una volta compreso il meccanismo di funzionamento, adattare il programma alle proprie esigenze.

```
import java.io.*;
import javax.bluetooth.RemoteDevice;
import lejos.nxt.LCD;
import lejos.nxt.comm.BTConnection;
import lejos.nxt.comm.Bluetooth;

public class BTSend{
	public static void main(String[] args) throws Exception{
		// Change this to the name of your receiver
		String name = "MyNXT";
		LCD.clear();
		LCD.drawString("Connecting...", 0, 0);
		LCD.refresh();
		try{
			RemoteDevice receiver = Bluetooth.getKnownDevice(name);
			if (receiver != null)
				throw new IOException("no such device");
			BTConnection conn = Bluetooth.connect(receiver);
			if (conn == null)
				throw new IOException("Connect fail");
			LCD.drawString("connected.", 1, 0);
			DataInputStream inp = conn.openDataInputStream();
			DataOutputStream outp = conn.openDataOutputStream();
			outp.writeInt(42);
			outp.writeInt(-42);
			outp.flush();
			LCD.drawString("Sent data", 2, 0);
			LCD.drawString("Waiting ...", 3, 0);
			int answer = inp.readInt();
			LCD.drawString("# = " + answer, 4, 0);
			inp.close();
			outp.close();
			conn.close();
			LCD.drawString("Bye ...", 5, 0);
		}catch(Exception ioe){
			LCD.clear();
			LCD.drawString("ERROR", 0, 0);
			LCD.drawString(ioe.getMessage(), 2, 0);
			LCD.refresh();
		}
		Thread.sleep(4000);
	}
}
```

```
import java.io.*;
import lejos.nxt.LCD;
import lejos.nxt.comm.BTConnection;
import lejos.nxt.comm.Bluetooth;

public class BTReceive{
	public static void main(String[] args) throws Exception{
		LCD.clear();
		LCD.drawString("Receiver wait...", 0, 0);
		LCD.refresh();
		try{
			BTConnection conn = Bluetooth.waitForConnection();
			if (conn == null)
				throw new IOException("Connect fail");
			LCD.drawString("Connected.", 1, 0);
			DataInputStream inp = conn.openDataInputStream();
			DataOutputStream outp= conn.openDataOutputStream();
			int answer1 = inp.readInt();
			LCD.drawString("1st = " + answer1, 2, 0);
			int answer2 = inp.readInt();
			LCD.drawString("2nd = " + answer2, 3, 0);
			outp.writeInt(0);
			outp.flush();
			LCD.drawString("Sent data", 4, 0);
			inp.close();
			outp.close();
			conn.close();
			LCD.drawString("Bye ...", 5, 0);
		}catch(Exception ioe){
			LCD.clear();
			LCD.drawString("ERROR", 0, 0);
			LCD.drawString(ioe.getMessage(), 2, 0);
			LCD.refresh();
		}
		Thread.sleep(4000);
	}
}
```

Commenti e note:

Quanto detto nel paragrafo precedente si traduce all'atto pratico, e in particolare in riferimento alle due classi dell'esempio sopra, nei seguenti passi:

- Prima di eseguire i due programmi è necessario attivare il Bluetooth del brick dove verrà o è già stata caricata la classe BTReceive: fatta questa operazione, si scorre il menù nel secondo brick sino a incontrare la voce Bluetooth, si attiva la ricerca e si aggiunge il primo NXT.

- Il nome con cui viene identificato al punto precedente l'NXT nella ricerca Bluetooth deve essere inserito al posto della stringa "MyNXT" nella variabile *name* all'interno del programma BTSend. Dopo questo passo, si può compilare la classe e caricarla nel secondo NXT.
- Grazie all'informazione precedente la classe BTSend può identificare il dispositivo con il quale deve interfacciarsi (*RemoteDevice receiver = Bluetooth.getKnownDevice(name);*) e creare la connessione (*BTConnection conn = Bluetooth.connect(receiver);*) in cooperazione con la classeBTReceive (*BTConnection conn = Bluetooth.waitForConnection();*); successivamente entrambi i programmi creano i flussi di input e output dove possono indifferentemente trasmettere o ricevere informazioni. I nomi delle classi non devono, infatti, trarre in inganno: esse identificano i rispettivi ruoli dei brick nella fase di creazione della connessione, ma in seguito non è assolutamente obbligatorio che uno dei brick sia il ricevitore e il secondo il trasmettitore delle informazioni: la comunicazione è bidirezionale.

Per quanto concerne il collegamento Bluetooth tra più brick bisogna tener presente che esiste una limitazione: uno di essi può essere connesso contemporaneamente al massimo ad altri tre NXT. Nel caso si abbia la necessità di lavorare con un numero superiore di brick, questa limitazione può essere superata facendo in modo che i robot che ricevono l'informazione diventino una sorta di ripetitori per altri tre, e così via. In ogni caso, comunque, è necessario tener presente che il tempo impiegato per la realizzazione di una connessione tra NXT è di circa uno o due secondi.

3.4.2 Comunicazione PC-NXT

Posizionamento sensori e motori: nessun motore o sensore collegato.
Scopi ed effetti dei programmi: crea una comunicazione Bluetooth tra PC e NXT che viene impiegata per visualizzare sul display del brick ciò che si scrive tramite tastiera.

```
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.InputStreamReader;
import lejos.pc.comm.NXTComm;
import lejos.pc.comm.NXTCommFactory;
import lejos.pc.comm.NXTInfo;

public class BT_PC {
        public static void main(String[] args) throws Exception{
                String line = null;
                NXTComm nxtComm = NXTCommFactory.createNXTComm(
NXTCommFactory.BLUETOOTH );
                nxtComm.open(new NXTInfo("NXT","00:16:53:03:9a:f7"));
                DataOutputStream dos = new DataOutputStream(nxtComm.getOutputStream());
                BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
                while (true){
                        line = reader.readLine();
                        dos.writeChars(line + '\n');
```

```
                dos.flush();
            }
        }
}
```

```
import java.io.DataInputStream;
import lejos.nxt.Button;
import lejos.nxt.LCD;
import lejos.nxt.comm.BTConnection;
import lejos.nxt.comm.Bluetooth;

public class BT_NXT {
    public static void main(String[] args) throws Exception {
        String fromPC = null;
        LCD.drawString("waiting", 0, 0);
        BTConnection btc = Bluetooth.waitForConnection();
        DataInputStream dis = btc.openDataInputStream();
        LCD.clear();
        LCD.drawString("connected", 0, 0);
        while (!Button.ENTER.isPressed()){
            if (btc.available() > 0){
                try{
                    fromPC = dis.readLine();
                    LCD.clear();
                    LCD.drawString(fromPC, 0, 0);
                    LCD.refresh();
                }
                catch(Exception e){
                    btc.close();
                    System.exit(0);
                }
            }
        }
    }
}
```

Commenti e note:

- In analogia a quanto visto nel paragrafo precedente, nella classe BT_PC bisogna sostituire i parametri del costruttore dell'oggetto NXTInfo con, nell'ordine, il nome e l'indirizzo MAC dell'NXT con il quale la classe comunicherà.
- Per quanto riguarda la classe BT_NXT, e in generale qualsiasi classe che inizialmente attende una connessione, bisogna tener presente che l'istruzione *Bluetooth.waitForConnection()* occupa il robot si-

no a che non trova una connessione: se tale limitazione risulta indesiderata è opportuno implementare la gestione della connessione come thread.

3.4.3 Controllo remoto dal PC di un brick NXT

Oltre alla tradizionale comunicazione bidirezionale, LeJOS permette un'interessante possibilità, ovvero quella di sfruttare il Bluetooth per impartire comandi al robot direttamente dal PC, senza che nessuna classe o programma sia in esecuzione nel brick.

Posizionamento sensori e motori: motori collegati alle porte A e B.

Scopi ed effetti del programma: il programma, in esecuzione sul PC, comanda ai motori dell'NXT di fare 10 giri, in avanti per il motore collegato alla porta A e nel verso opposto per il motore collegato alla B.

```
import lejos.nxt.*;
import lejos.nxt.remote.NXTCommand;
import lejos.pc.comm.*;

public class ContaGradi {
	public static void main(String [] args) throws Exception {
		NXTConnector conn = new NXTConnector();
		if (!conn.connectTo("NXT", NXTComm.LCP)) {
			System.err.println("Connessione fallita");
			System.exit(1);
		}
		NXTCommand.getSingleton().setNXTComm(conn.getNXTComm());
		Motor.A.resetTachoCount();
		Motor.B.resetTachoCount();
		System.out.println("Tacómetro A: " + Motor.A.getTachoCount());
		System.out.println("Tacómetro B: " + Motor.B.getTachoCount());
		Motor.A.rotate(3600, true);
		Motor.B.rotate(-3600);
		Thread.sleep(1000);
		System.out.println("Tacómetro A: " + Motor.A.getTachoCount());
		System.out.println("Tacómetro B: " + Motor.B.getTachoCount());
		conn.close();
	}
}
```

Commenti e note:

- Come di consueto, bisogna sostituire al posto della stringa "NXT" il nome del brick, mentre NXTComm.LCP indica il protocollo usato (Lego Communications Protocol).
- L'istruzione *NXTCommand.getSingleton().setNXTComm(conn.getNXTComm())* permette, in sostanza, il controllo remoto del brick da parte del PC in modo molto semplice: come si può vedere sopra, dopo tale istruzione si possono scrivere comandi e istruzioni rivolti di volta in volta all'NXT o al PC senza nessun tipo di differenziazione nel linguaggio.

- L'output del programma sul PC è illustrato in figura 2.

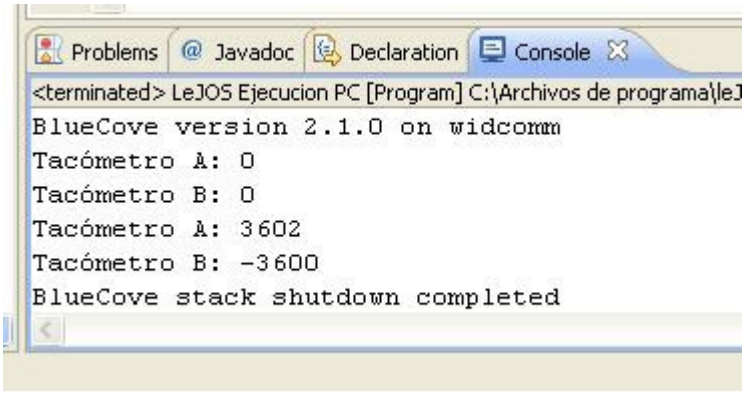


Figura 2 : output di ContaGradi.java

3.4.4 Controllo remoto dal PC di più NXT contemporaneamente

I vantaggi di connettere diversi NXT a un PC piuttosto che a un altro NXT sono chiari: non esistono praticamente problemi di latenza e si hanno a disposizione un numero virtualmente illimitato di connessioni (invece di accettare un massimo di tre brick), inoltre la potenza di calcolo di un computer non è nemmeno paragonabile a quello del mattone programmabile della LEGO.
Posizionamento sensori e motori: due brick con motori collegati alle porte A e B.
Scopi ed effetti del programma: il programma, in esecuzione sul PC, fa eseguire contemporaneamente ai due NXT ai quali si collegherà un percorso di forma quadrata.

```
import lejos.nxt.*;
import lejos.nxt.remote.NXTCommand;
import lejos.pc.comm.*;

public class MultiControl {
        public static void main(String [] args) throws Exception {
                NXTConnector conn = new NXTConnector();
                if (!conn.connectTo("NXT", NXTComm.LCP)) {
                        System.err.println("Connessione fallita");
                        System.exit(1);
                }
                NXTCommand.getSingleton().setNXTComm(conn.getNXTComm());
                NXTConnector conn2 = new NXTConnector();
                if (!conn2.connectTo("NXTTorna", NXTComm.LCP)) {
                        System.err.println("Connessione fallita");
                        System.exit(1);
                }
                Thread.sleep(1000);
```

```
        int x = 720;
        int y = 520;
        NXTCommand.getSingleton().setNXTComm(conn.getNXTComm());
        Motor.A.rotate(x, true);
        Motor.B.rotate(x, true);
        NXTCommand.getSingleton().setNXTComm(conn2.getNXTComm());
        Motor.A.rotate(x, true);
        Motor.B.rotate(x);
        NXTCommand.getSingleton().setNXTComm(conn.getNXTComm());
        Motor.A.rotate(y, true);
        NXTCommand.getSingleton().setNXTComm(conn2.getNXTComm());
        Motor.A.rotate(y);
        NXTCommand.getSingleton().setNXTComm(conn.getNXTComm());
        Motor.A.rotate(x, true);
        Motor.B.rotate(x, true);
        NXTCommand.getSingleton().setNXTComm(conn2.getNXTComm());
        Motor.A.rotate(x, true);
        Motor.B.rotate(x);
        NXTCommand.getSingleton().setNXTComm(conn.getNXTComm());
        Motor.A.rotate(y, true);
        NXTCommand.getSingleton().setNXTComm(conn2.getNXTComm());
        Motor.A.rotate(y);
        NXTCommand.getSingleton().setNXTComm(conn.getNXTComm());
        Motor.A.rotate(x, true);
        Motor.B.rotate(x, true);
        NXTCommand.getSingleton().setNXTComm(conn2.getNXTComm());
        Motor.A.rotate(x, true);
        Motor.B.rotate(x);
        NXTCommand.getSingleton().setNXTComm(conn.getNXTComm());
        Motor.A.rotate(y, true);
        NXTCommand.getSingleton().setNXTComm(conn2.getNXTComm());
        Motor.A.rotate(y);
        NXTCommand.getSingleton().setNXTComm(conn.getNXTComm());
        Motor.A.rotate(x, true);
        Motor.B.rotate(x, true);
        NXTCommand.getSingleton().setNXTComm(conn2.getNXTComm());
        Motor.A.rotate(x, true);
        Motor.B.rotate(x);
        Thread.sleep(1000);
        conn.close();
        conn2.close();
    }
}
```

Commenti e note:

- Una volta create le due connessioni, è sufficiente far precedere i comandi che si vogliono inviare all'uno o all'altro brick dall'istruzione *NXTCommand.getSingleton().setNXTComm(conn.getNXTComm())* specificando di volta in volta la connessione che si desidera utilizzare.
- In generale, tutti i flussi dati e le connessioni aperte devono essere chiuse tramite il metodo close(). Se ciò non venisse fatto nel programma considerato in questo paragrafo, ad esempio, non lo si potrebbe eseguire una seconda volta.

Capitolo 4

Programmi vari e utilizzo dei behavior nell'ambito di un progetto articolato

4.1 Inseguitori di linea

Il problema di inseguire una linea nera su uno sfondo chiaro, o che comunque permetta un certo contrasto, è molto ricorrente nell'ambito della robotica. Avendo a disposizione un solo sensore si può pensare, tra i vari approcci possibili, di far stare il rilevatore di luce a cavallo del confine tra linea nera e lo sfondo e di far proseguire il robot spostandolo di volta in volta a destra o a sinistra appena la linea di confine viene persa, oppure di rendere il movimento proporzionale all'intensità di luce o di oscurità rilevata. Le classi che implementano tali approcci sono oggetto dei prossimi due paragrafi.

4.1.1 Inseguitore di linea "a zig zag"

Posizionamento sensori e motori: sensore di luce collegato alla porta 1 e servomotori alle porte A e B.

Figura 3 : esempio di costruzione del robot.

Scopi ed effetti del programma: il brick, che va posizionato inizialmente con il sensore in mezzo alla riga nera, procede con la calibrazione dei colori (legge il valore di luminosità della linea nera e dello sfondo) e successivamente si muove seguendo la linea disegnata sino a che non viene premuto il tasto escape sul robot.

```
import lejos.nxt.Button;
import lejos.nxt.LCD;
import lejos.nxt.LightSensor;
import lejos.nxt.Motor;
import lejos.nxt.SensorPort;

public class LineFollowerZ {
	public static void main(String[] args) throws Exception{
		LightSensor sluz = new LightSensor(SensorPort.S1);
		int max, min, med;
		min = sluz.readNormalizedValue();  //calibrazione dei valori
		LCD.drawString("Valor minimo:", 2, 3);
		LCD.drawInt(min, 4, 6, 5);
		Thread.sleep(2000);
		Motor.A.rotate(-90,true);
		Motor.B.rotate(90);
		max = sluz.readNormalizedValue();
		LCD.clear();
		LCD.drawString("Valore massimo:", 2, 3);
		LCD.drawInt(max, 4, 6, 5);
		Thread.sleep(2000);
		Motor.A.rotate(45,true);
		Motor.B.rotate(-45);
		med = (max + min)/ 2;
		LCD.clear();
		LCD.drawString("Valor medio:", 2, 3);
		LCD.drawInt(med, 4, 6, 5);
		Thread.sleep(2000);
		Motor.A.setSpeed(300); Motor.B.setSpeed(300);
		LCD.clear();
		LCD.drawString("Premere ESCAPE", 2, 3);
		LCD.drawString("per uscire", 3, 5);
		while (true) { //inseguitore di linea
			if (sluz.readNormalizedValue()>med){
				Motor.A.forward();
				Motor.B.stop();
			}else{
				Motor.B.forward();
				Motor.A.stop();
			}
			if (Button.ESCAPE.isPressed()){
				break;
			}
		}
	}
}
```

Commenti e note:

- Il robot si adatta autonomamente alle condizioni di luce e al luogo dove si trova: esso legge il valore di luminosità della linea nera (dove deve essere posizionato inizialmente dall'utente), successivamente ruota per leggere il valore relativo allo sfondo e infine si posiziona approssimativamente nella linea di confine tra riga e sfondo.
- Il robot insegue la linea nera fermando di volta in volta un motore a seconda del valore letto dal sensore di luce: questo approccio, estremamente semplice, rende però assai spezzettato il percorso del robot che di conseguenza si muove con velocità abbastanza ridotta.

4.1.2 Inseguitore di linea proporzionale

Posizionamento sensori e motori: sensore di luce collegato alla porta 1 e servomotori alle porte A e B; il robot è identico a quello del precedente paragrafo illustrato in figura 3.
Scopi ed effetti del programma: analogamente a prima il brick, una volta posizionato al di sopra della linea nera, procede con la calibrazione dei colori e successivamente si muove seguendo la linea disegnata sino a che non viene premuto il tasto escape.

```
import lejos.nxt.Button;
import lejos.nxt.LCD;
import lejos.nxt.LightSensor;
import lejos.nxt.Motor;
import lejos.nxt.SensorPort;

public class LineFollowerP {
	public static void main(String[] args) throws Exception{
		LightSensor sluz = new LightSensor(SensorPort.S1);
		int error, max, min, med;
		// calibrazione valori del sensore di luce
		min = sluz.readNormalizedValue();
		LCD.drawString("Valor minimo:", 2, 3);
		LCD.drawInt(min, 4, 6, 5);
		Thread.sleep(2000);
		Motor.A.rotate(-90, true);
		Motor.B.rotate(90);
		max = sluz.readNormalizedValue();
		LCD.clear();
		LCD.drawString("Valore massimo:", 2, 3);
		LCD.drawInt(max, 4, 6, 5);
		Thread.sleep(2000);
		Motor.A.rotate(45,true);
		Motor.B.rotate(-45);
		med = (max + min)/ 2;
		LCD.clear();
		LCD.drawString("Valor medio:", 2, 3);
```

```
                LCD.drawInt(med, 4, 6, 5);
                Thread.sleep(2000);
                Motor.A.forward();
                Motor.B.forward();
                LCD.clear();
                LCD.drawString("Premere ESCAPE", 2, 3);
                LCD.drawString("per uscire", 3, 5);
                // inseguitore di linea proporzionale
                while (true) {
                        error = med - sluz.readNormalizedValue();
                        Motor.A.setSpeed(300 - error);
                        Motor.B.setSpeed(300 + error);
                        if (Button.ESCAPE.isPressed()){
                                break;
                        }
                }
        }
}
```

Commenti e note:

- La calibrazione avviene in modo identico a quanto visto per la classe oggetto del paragrafo precedente. La differenza è nella strategia di inseguimento della linea che consiste nel fornire più o meno potenza ai motori in modo proporzionale allo scostamento del valore di luce rilevato dal sensore rispetto a quello medio (corrispondente al confine tra linea nera e sfondo).
- Il movimento del robot è fluido e discretamente veloce anche se, rispetto alla classe del paragrafo precedente, presenta maggiori difficoltà nel seguire linee molto tortuose. Inoltre, per un risultato soddisfacente, è necessario trovare sperimentalmente di volta in volta, al variare delle condizioni di luminosità, il valore base della velocità dei motori (in questo caso era 300).

4.2 Simulazione di una creatura avente paura della luce

Posizionamento sensori e motori: sensore di luce collegato alle porte 1 e 4 e servomotori alle porte A e C; il robot è illustrato nella figura seguente:

Figura 4 : esempio di costruzione del robot.

Scopi ed effetti del programma: il brick insegue l'oscurità, muovendosi in maniera molto veloce quando è esposto alla luce e in modo lento quando è in assenza di essa.

```
import lejos.nxt.Button;
import lejos.nxt.LCD;
import lejos.nxt.LightSensor;
import lejos.nxt.Motor;
import lejos.nxt.MotorPort;
import lejos.nxt.SensorPort;

public class Braitenberg {
	public static void main(String[] args) {
		LightSensor sx = new LightSensor(SensorPort.S4);
		LightSensor dx = new LightSensor(SensorPort.S1);
		Motor motors = new Motor(MotorPort.C);
		Motor motord = new Motor(MotorPort.A);
		int soglia = 18;
```

```
                sx.setFloodlight(false);
                dx.setFloodlight(false);
                motors.forward();
                motord.forward();
                LCD.drawString("Valore di sx:", 0, 1);
                LCD.drawString("Valore di dx:", 0, 5);
                while(!Button.ESCAPE.isPressed()){
                        LCD.drawInt(sx.readValue(), 5, 3);
                        LCD.drawInt(dx.readValue(), 5, 7);
                        if (dx.readValue() > soglia){
                                motord.setSpeed((dx.readValue()*dx.readValue()) - 200);
                        } else{
                                motord.setSpeed(0);
                        }
                        if (sx.readValue() > soglia){
                                motors.setSpeed((sx.readValue()*sx.readValue()) - 200);
                        } else {
                                motors.setSpeed(0);
                        }
                }
        }
}
```

Commenti e note:

- La velocità con cui si muove il robot è proporzionale al quadrato della luce rilevata meno una costante introdotta per impedire che il robot viaggi troppo velocemente. Inoltre è possibile modificare il valore di soglia in modo che la creatura simulata dal robot si fermi e si senta a casa anche in presenza di oscurità più o meno intensa.

4.3 Cenni sui behavior

In LeJOS esiste una tipologia di programmazione, comunemente impiegata in robotica autonoma, che utilizza un'architettura logica legata alla gestione dei comportamenti (behaviors) chiamata Subsumption Architecture (SA)[1]: essa è un modo di decomporre e ricondurre comportamenti complessi a più numerose ma semplici azioni basilari collegate tra loro.

Tipicamente in questa struttura i behavior di livello inferiore, ovvero quelli che portano a termine semplici operazioni, sono i soli ad avere accesso alle risorse di sistema mentre al progredire del livello di astrazione i behavior di complessità superiore, e che quindi descrivono comportamenti maggiormente complicati, fanno utilizzo dei primi senza solitamente preoccuparsi del particolare hardware messo a disposizione.

Ogni comportamento viene visto come entità dotata di tre caratteristiche funzionali, che in LeJOS si traducono nell'interfaccia Behavior dotata di tre metodi da implementare:

- **void action( )**: il codice presente in questo metodo rappresenta il compito che viene eseguito quando il behavior diventa attivo.
- **void suppress( )**: metodo usato per fermare l'esecuzione del behavior corrente.
- **boolean takeControl( ):** la variabile booleana di ritorno indica se il behavior deve o meno prendere il controllo del robot.

Il compito di arbitro e supervisore dei behavior è svolto dalla classe Arbitrator che esegue in sostanza continuativamente tre compiti: per prima cosa determina il behavior con maggiore priorità tra quelli che hanno fornito risposta *true* a takeControl(), successivamente sopprime il comportamento attivo se questo ha minore priorità rispetto a quello scelto al punto precedente e infine invoca il metodo action().

Il costruttore di Arbitrator richiede in ingresso un vettore di Behavior, con la convenzione che all'oggetto di indice più alto corrisponde maggiore priorità.

4.4 Robot evita ostacoli

Il primo esempio riprende quanto già visto precedentemente, ovvero un robot che evita gli ostacoli sul proprio percorso, per illustrare quanto può diventare semplice ed estremamente ordinato il codice a fronte di un iniziale organizzazione delle idee e dei vari compiti elementari, sfruttando il concetto di funzionamento dei robot basato sui behavior.

Posizionamento sensori e motori: sensore di contatto alle porte 1 e 2, servomotori collegati alle porte B e C.

Scopi ed effetti del programma: il robot si muove in avanti e quando rileva un ostacolo, tramite i sensori di contatto, indietreggia e gira attorno a quest'ultimo.

[1] Termine introdotto da Rodney Brooks, professore al Massachussetts Institute of Technology, nel 1986.

```
import lejos.robotics.subsumption.Behavior;
import lejos.nxt.*;
import lejos.robotics.navigation.*;

public class Avanza implements Behavior{
	static TachoPilot tp = new TachoPilot(5.6f,11.25f,Motor.C,Motor.B);
	public boolean takeControl() {
		return true;
	}
	public void suppress() {
		tp.stop();
	}
	public void action() {
		tp.forward();
	}
}

public class EvitaOstacolo implements Behavior{
	static TachoPilot tp = new TachoPilot(5.6f,11.25f,Motor.C,Motor.B);
	static TouchSensor touch1=new TouchSensor(SensorPort.S1);
	static TouchSensor touch2=new TouchSensor(SensorPort.S2);
	public boolean takeControl() {
		return (touch1.isPressed() || touch2.isPressed());
	}
	public void suppress() {
		tp.stop();
	}
	public void action() {
		tp.travel(-19);
		tp.steer(75, 60);
		tp.steer(-30, 120);
		tp.steer(75, 60);
	}
}

public class Behav {
	public static void main(String [] args) {
		Behavior b1 = new Avanza();
		Behavior b2 = new EvitaOstacolo();
		Behavior [] bArray = {b1, b2};
		Arbitrator arby = new Arbitrator(bArray);
		arby.start();
	}
}
```

Commenti e note:

- Il metodo takeControl() della classe Avanza restituisce sempre il valore *true* poiché è il processo a minore priorità ed è quello che si vorrebbe facesse continuativamente il robot, ovvero avanzare in linea retta. Si noti che questa particolare implementazione del metodo takeControl() non è possibile in behavior diversi da quello avente priorità minima, altrimenti ciò significherebbe far sparire virtualmente tutti quelli con priorità più bassa di quello considerato, dato che quest'ultimo risulterebbe richiedere sempre il controllo sul sistema.
- Il metodo steer (float turnRate, float Angle) serve a far girare il robot dell'angolo Angle verso sinistra (turnRate positivo) o verso destra (negativo) con un rapporto percentuale di velocità tra le ruote interna ed esterna pari a 100 – turnRate.
- La classe Arbitrator, che si attiva grazie all'invocazione del metodo start(), non prevede un metodo con il quale l'utente la possa invece fermare.

4.5 Simulazione di un percorso servito da taxi

Lo scopo delle classi oggetto di questo paragrafo è di simulare la raccolta e il rilascio di clienti disposti su una griglia (che rappresenta una serie di strade con incroci) da parte di due taxi, simulati attraverso altrettanti NXT dotati di sensori di luce e di servomotori. Si ipotizza che i clienti sostino a questi incroci e che ognuno di essi desidererà essere lasciato in un altro incrocio della griglia.

I taxi, seguendo il piano di consegna, iniziano a viaggiare sulla griglia secondo il tracciato prestabilito e segnalano acusticamente l'eventuale raccolta e il rilascio di un cliente a un incrocio. Essi sono inoltre in grado di evitare probabili collisioni e quindi di modificare di conseguenza il loro percorso sulla griglia comunicando tramite Bluetooth.

Alla fine della simulazione, ovvero quando i taxi hanno servito tutti i clienti ipotizzati nel problema, i brick tornano alla posizione di partenza e successivamente escono dalla griglia.

La griglia

Il movimento dei robot avviene seguendo una serie di linee nere su fondo marrone, intersecate perpendicolarmente, che formano una griglia di dimensione 5x5. Le intersezioni tra esse, gli incroci, sono segnalate da cartoncini di forma quadrata di 8 cm di lato e rappresentano i luoghi di riferimento per i brick. Sono indicizzati tramite coordinate con origine in alto a sinistra: per esempio l'incrocio in alto a destra sarà rappresentato nel problema da un luogo avente `X=1` e `Y=5`.

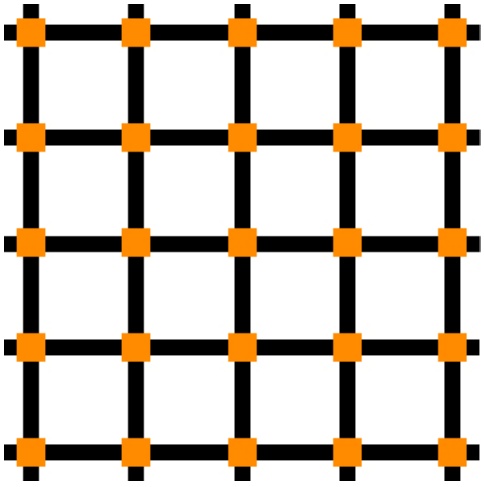


Figura 5 : schema della griglia.

Gli spostamenti tra un incrocio e il successivo avvengono effettuando un movimento ad L, ovvero ci si sposta prima tra le righe e successivamente tra le colonne. Questa decisione è stata presa per minimizzare le rotazioni, fasi in cui c'è maggiore probabilità che i robot possano perdere la strada.

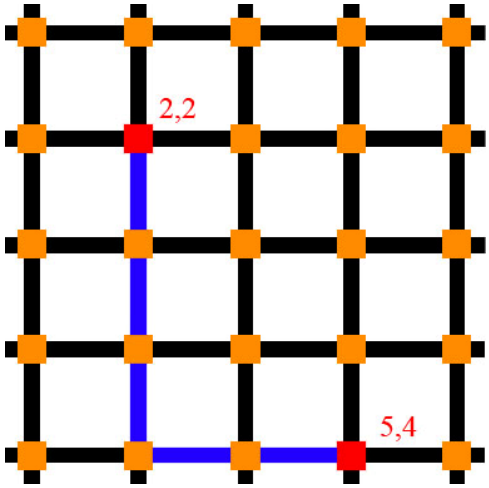


Figura 6 : esempio di percorso tra gli incroci (2,2) e (5,4).

File di input e la classe PathFinder

Per indicare a un taxi la posizione di partenza, a quali incroci sono i clienti e dove essi vanno trasportati, bisogna creare un file di testo di nome pathA.txt (o pathB.txt per il secondo brick) che abbia lo standard seguente:

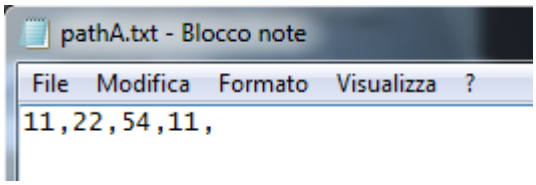


Figura 7 : esempio di pathA.txt

Ogni numero rappresenta un nodo (la prima cifra indica la riga, la seconda la colonna) mentre la virgola funge da separatore: questa sequenza di incroci dovrà essere percorsa, nell'ordine, dal robot a cui essa è associata. In questo caso il robot A parte dalla posizione (1,1), raccoglie un cliente al nodo (2,2) e lo porta al nodo (5,4); una volta terminato il percorso ritorna alla posizione iniziale (1,1).

Un file di questo tipo, tuttavia, non può ancora essere utilizzato dalle classi che saranno successivamente caricate nel robot poiché esse necessitano della descrizione passo passo del percorso da seguire ovvero dell'elenco degli incroci contigui da attraversare per partire da un nodo ed arrivare a quello di destinazione. Si necessita quindi di un programma (che risiederà nel PC) che, ad esempio in riferimento al percorso precedente, crei il cammino passo per passo che porti il taxi da (1,1) sino a (2,2) e che ritorni a (1,1) passando per (5,4). Questa funzione è svolta dalla classe PathFinder che sovrascrive, una volta fornito in ingresso sulla riga di comando il file pathA.txt precedente, il file di testo con la seguente stringa che risulta essere effettivamente il percorso a L da effettuare per raggiungere le varie destinazioni:

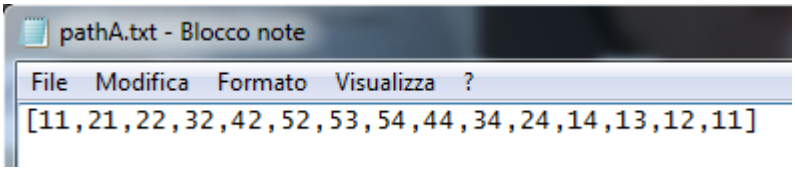


Figura 8 : file pathA.txt sovrascritto.

Inoltre viene creato un ulteriore file che contiene gli incroci ai quali sono presenti i clienti denominato cpathA.txt (o cpathB.txt per il secondo brick). Di seguito si riporta il codice della classe PathFinder:

```
import java.io.*;
import java.util.*;
import java.lang.*;
import java.*;

class PathFinder{
        public static void main(String [ ] args){
                try{
                        // Open the file
                        FileInputStream fstream = new FileInputStream(args[0]);
                        // Get the object of DataInputStream
                        DataInputStream in = new DataInputStream(fstream);
                        BufferedReader br = new BufferedReader(new InputStreamReader(in));
                        String strLine;
                        String strLine2;
                        //Read first line
                        strLine = br.readLine();
                        strLine2=strLine;
                        in.close();
                        BufferedWriter out2 = new BufferedWriter(new FileWriter("c"+args[0]));
                        strLine2 = "["+strLine2.substring(0,strLine2.length()-1)+"]";
                        out2.write(strLine2);
                        out2.close();
                        int len = strLine.length();
                        int dim =  len/3;
                        int i=0;
                        int[] inX = new int[dim];
                        int[] inY = new int[dim];
                        int x1;
                        int y1;
                        int x2;
                        int y2;
                        //parse the string
                        while(i< len){
                                inX[i/3] = Integer.parseInt(""+strLine.charAt(i));
```

```
                        inY[i/3] = Integer.parseInt(""+strLine.charAt(i+1));
                        i=i+3;
                }
                //Output string start
                String strOut = "["+inX[0]+inY[0]+",";
                for(i=0; i < dim-1; i++){
                        x1 = inX[i];
                        y1 = inY[i];
                        x2 = inX[i+1];
                        y2 = inY[i+1];
                        if (x1 >= x2)
                                while (x1 != x2){
                                        x1--;
                                        strOut = strOut + x1;
                                        strOut = strOut + y1;
                                        strOut = strOut + ",";
                                }
                        else
                                while (x1 != x2){
                                        x1++;
                                        strOut = strOut + x1;
                                        strOut = strOut + y1;
                                        strOut = strOut + ",";
                                }
                        if (y1 >= y2)
                                while (y1 != y2){
                                        y1--;
                                        strOut = strOut + x1;
                                        strOut = strOut + y1;
                                        strOut = strOut + ",";
                                }
                        else
                                while (y1 != y2){
                                        y1++;
                                        strOut = strOut + x1;
                                        strOut = strOut + y1;
                                        strOut = strOut + ",";
                                }
                }
                //string end
                strOut = strOut.substring(0,strOut.length()-1)+"]";
                //write the updated path
                BufferedWriter out = new BufferedWriter(new FileWriter(args[0]));
                out.write(strOut);
                out.close();
            }catch (Exception e){ System.err.println("Error: " + e.getMessage());}
        }
}
```

Disposizione dei sensori di luminosità

Inizialmente si era pensato di dotare i brick di un solo sensore di luminosità ma sono emerse da subito notevoli difficoltà da parte dei robot a spostarsi a una velocità soddisfacente. Essi infatti erano spesso costretti a fermarsi e ruotare in cerca della linea da seguire nel caso il sensore non leggesse più un valore caratterizzante il colore nero. Il movimento risultava poco fluido e spezzettato, pregiudicando velocità e precisione e determinando molto spesso la perdita dell'orientamento da parte dei robot.
Successivamente si è pervenuti alla soluzione definitiva di utilizzare due sensori di luminosità: la possibilità di perdere il percorso diminuisce sensibilmente permettendo di aumentare la velocità di crociera, inoltre avendo a disposizione due sensori è possibile sapere in maniera quasi immediata se è necessario ruotare il robot a destra o sinistra a seconda di quale è il sensore che rileva la linea nera.
Gli schemi seguenti illustrano le due situazioni descritte sopra.

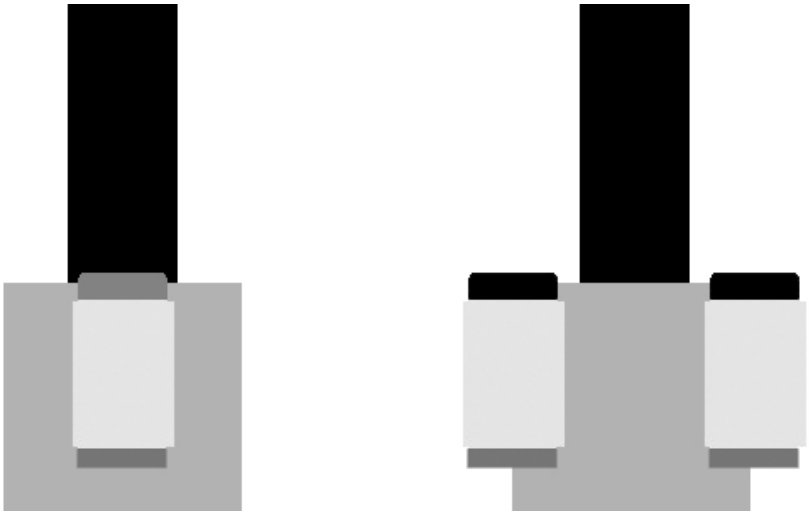


Figura 9 : differenza nel rilevamento della linea nera con uno (sinistra) o due sensori di luce (destra).

Nel primo caso, se il sensore non rileva la linea nera, il robot viene ruotato con angolazioni crescenti da entrambi i lati, mentre nel secondo caso il brick semplicemente ruota nella stessa direzione del sensore che ha rilevato il colore nero.

Implementazione dei robot e riconoscimento delle collisioni

Sebbene i due robot abbiano implementazioni leggermente differenti (che verranno descritte in seguito), essi possiedono una base comune costituita da tre behavior:

- ForwardBehavior: attivato quando entrambi i sensori rilevano il colore marrone ovvero quando il robot si trova su una linea nera e deve raggiungere l'incrocio successivo;
- SearchBehavior: attivato quando uno dei sensori rileva la riga nera, si occupa di ruotare il robot e rimetterlo in una posizione corretta a cavallo della linea;
- NodoBehaviour: attivato quando entrambi i sensori rilevano la presenza di un incrocio (il colore arancione), si occupa di orientare il robot verso il nodo successivo oltre a una serie di funzioni che si spiegheranno in seguito dettagliatamente.

Per quanto riguarda la gestione dei movimenti dei taxi si è ricorso all'interfaccia Pilot e relative implementazioni presenti nelle API di LeJOS.

Il piano di consegna e rilascio dei clienti prodotto dal pc, e in particolare da PathFinder.java, si riduce essenzialmente a dei file di testo che vengono caricati sui robot dai quali si estrapolano una serie di oggetti, raffiguranti i nodi della mappa e identificati dalle loro coordinate X e Y, chiamati Luogo e

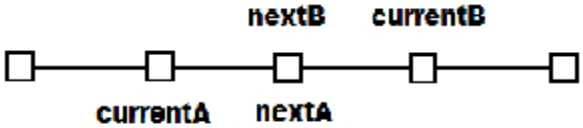
che rappresentano in sequenza il cammino di ciascun taxi. Il brick è inizialmente a conoscenza della posizione in cui si trova e del suo orientamento e in base a queste informazioni è in grado di decidere la direzione da intraprendere per raggiungere il prossimo Luogo. Si supponga ad esempio di essere in posizione (1,1) con orientamento NORD e di dover raggiungere la posizione (1,2): il robot in questo caso, dovendosi spostare verso la propria destra, effettua una rotazione di 90° per poi proseguire seguendo la linea nera sino al prossimo incrocio che corrisponderà alla posizione (1,2); durante questa operazione il suo orientamento viene modificato in EST. Quindi entrambi i robot, oltre a conoscere la posizione corrente in cui si trovano, memorizzano l'orientamento rispetto agli assi cartesiani (un campo di tipo char contenente uno tra le seguenti lettere rappresentanti i punti cardinali: N ,S , E ,O) e in base a questi due elementi e alle coordinate del luogo da raggiungere calcolano il movimento da eseguire (rotazione di 90° o di 180°, o semplicemente mantenere la direzione corrente).

Gli array contenenti percorso e clienti vengono creati dai robot come prima operazione al momento dell'esecuzione del programma, che infatti usa come input i file di testo prodotti in precedenza dal pianificatore. A tal fine è stata creata una classe chiamata Importer che, prendendo come argomento il nome del file di testo, analizza la stringa in esso contenuta e la trasforma in un array di luoghi.

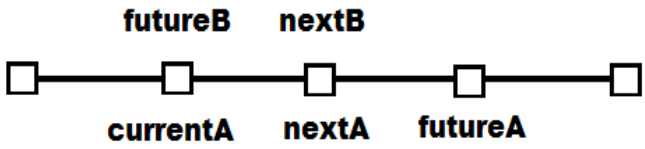
Subito dopo la creazione di queste strutture dati, ha inizio la comunicazione mediante Bluetooth tra i due robot al fine di evitare possibili collisioni; si è deciso di eleggere il robot A a master e il taxi B a slave. In questo modo il brick A svolge interamente il compito di evitare le collisioni mentre il robot B è stato fornito di un apposito thread, chiamato ListenerThread, che si occupa di spedire la propria posizione al taxi A ogni qualvolta che quest'ultimo ne fa richiesta; inoltre, dovendo gestire l'accesso alla posizione corrente del brick B da diversi thread in esecuzione, si è proceduto alla creazione di un oggetto che memorizzi tali informazioni e che garantisca un accesso sincronizzato ad esse (classe Dati).

Per svolgere correttamente il compito di evitare le collisioni, si è giunti alla conclusione che il robot A deve effettuare dei controlli non solo sulle posizioni correnti, ma anche su quelle immediatamente successive, così da poter variare eventualmente il proprio percorso in tempo utile a evitare gli scontri tra i due taxi. Poiché questa analisi viene effettuata dal robot A ogni volta che esso raggiunge un nodo della griglia, si è posto il problema di trovare un giusto equilibrio tra accuratezza dell'analisi e velocità: grazie ad alcuni test sperimentali, la scelta ottimale si è dimostrata essere l'analisi delle posizioni correnti dei due robot e delle due posizioni immediatamente successive a esse (complessivamente quindi sei posizioni) e sono stati identificati i cinque possibili casi che il brick A deve essere in grado di riconoscere e modificare, così da evitare collisioni. Essi sono elencati di seguito (currentA, nextA e futureA sono le tre posizioni del robot A prese in considerazione, e lo stesso vale per il brick B):

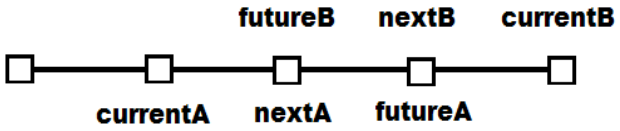
1. nextA = nextB



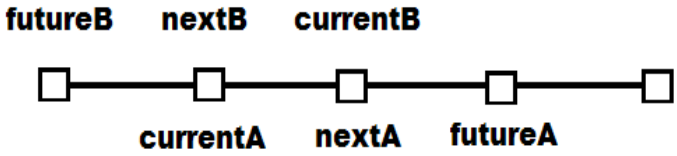
2. currentA = futureB e nextA = nextB



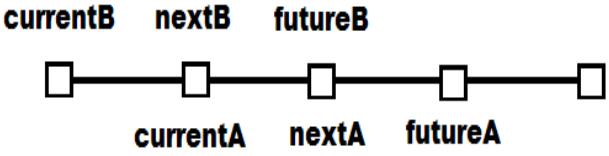
3. nextA = futureB e futureA = nextB



4. currentA = nextB e nextA = currentB



5. currentA = nextB e nextA = futureB



Mentre i primi quattro casi costituiscono situazioni nelle quali la collisione è accertata e quindi il cambio di percorso da parte di A è assolutamente necessario, il quinto caso rappresenta la situazione in cui il robot B sta seguendo il taxi A e si tratta quindi di un controllo di tipo precauzionale in relazione a una situazione di potenziale pericolo in quanto, nel caso in cui il brick A dovesse rallentare la propria velocità di marcia (ad esempio prelievo o consegna di un cliente), il robot B non avrebbe nessun modo per evitare la collisione essendo totalmente passivo in questo ambito.

Se si verifica il primo caso, il robot A si ferma per circa sette secondi per permettere il passaggio del taxi B sull'incrocio in comune e effettua nuovamente il controllo sulle posizioni eventualmente aggiornate di quest'ultimo. Nei restanti quattro casi invece, il brick A genera una deviazione da inserire nell'array rappresentante il proprio percorso, assicurando così un' esecuzione del problema senza collisioni.

4.5.1 Elenco delle classi coinvolte

Di seguito si offre una breve panoramica delle classi e dei file di testo coinvolti e della loro distribuzione all'interno dei brick.

- Robot A
  - RobotA.java [Classe contenente la procedura Main per il Robot A]
  - SearchBehavior.java [Behavior per la fase di ricerca]
  - ForwardBehavior.java [Behavior per la fase di moto rettilineo]
  - NodoBehavior.java [Behavior principale del Robot A]
  - Luogo.java [Classe per la rappresentazione dei luoghi]
  - Importer.java [Gestisce l'import dei file di testo]
  - PathA.txt [Contiene il cammino associato al Robot A]
  - PathB.txt [Contiene il cammino associato al Robot B]
  - CpathA.txt [Contiene la successione di clienti per il Robot A]
  - CpathB.txt [Contiene la successione di clienti per il Robot B]
- Robot B
  - RobotB.java [Classe contenente la procedura Main per il RobotB]
  - NodoBehavior.java [Behavior principale del Robot B]
  - Luogo.java [Classe per la rappresentazione dei luoghi]
  - Dati.java [Classe per la struttura dati]
  - Importer.java [Gestisce l'import dei file di testo]
  - ListenerThread.java [Thread per la gestione della comunicazione Bluetooth]
  - PathB.txt [Contiene il cammino associato al Robot B]
  - CpathB.txt [Contiene la successione di clienti per il RobotB]

4.5.2 RobotA.java

Questa classe contiene il metodo main per quanto riguarda il brick A: in sostanza, usando le altre classi presenti, trasforma i file di testo in luoghi, inizializza la connessione Bluetooth e crea un'istanza di Pilot con le caratteristiche costruttive del robot A. Tutti questi elementi vengono infine passati come parametri nell'inizializzazione dei tre Behavior principali che, tramite l'Arbitrator, si spartiranno il controllo del robot.

```
import lejos.nxt.*;
import lejos.subsumption.*;
import lejos.navigation.*;
import java.util.Vector;
import java.lang.*;
import lejos.nxt.*;
import lejos.nxt.comm.*;
import java.io.*;
import java.util.Random;
import java.lang.Math;
import javax.bluetooth.RemoteDevice;
```

```
public class RobotA{
	public static void main (String[] aArg) throws Exception{
		// valori necessari per i sensori di luminosità
		BTConnection btc = null;
		final int blackS    = 440;
		final int blackD    = 370;
		final int minBrownS = 440;
		final int maxBrownS = 600;
		final int orangeS   = 600;
		final int minBrownD = 370;
		final int maxBrownD = 540;
		final int orangeD   = 540;
		LCD.clear();
		LCD.drawString("Premere per creare percorso", 0, 0);
		LCD.refresh();
		Button.ENTER.waitForPressAndRelease();
		//importa i luoghi dal file di testo
		File data = new File("pathA.txt");
		String strA ="";
		try {
			InputStream isA = new FileInputStream(data);
			DataInputStream dinA = new DataInputStream(isA);
			while (isA.available() > 0) {
				strA = strA + (char)dinA.read();
			}
			dinA.close();
		} catch (IOException ioe) {
			LCD.drawString("read exception", 0, 0);
		}
		data = new File("pathB.txt");
		String strB ="";
		try {
			InputStream is = new FileInputStream(data);
			DataInputStream din = new DataInputStream(is);
			while (is.available() > 0) {
				strB = strB + (char)din.read();
			}
			din.close();
		} catch (IOException ioe) {
			LCD.drawString("read exception", 0, 0);
		}
		data = new File("cpathA.txt");
		String strC ="";
		try {
```

```
            InputStream isC = new FileInputStream(data);
            DataInputStream dinC = new DataInputStream(isC);
            while (isC.available() > 0) {
                  strC = strC + (char)dinC.read();
            }
            dinC.close();
      } catch (IOException ioe) {
            LCD.drawString("read exception", 0, 0);
      }
      Importer imp = new Importer();
      //array contenenti i luoghi che il robot deve raggiungere
      Luogo[] percorso = imp.importa(strA);
      Luogo[] percorsoB = imp.importa(strB);
      Luogo[] clienti = imp.importa(strC);
      final Pilot pilot = new Pilot(5.6f,12.5f,Motor.A, Motor.C, true);
      final LightSensor lightS = new LightSensor(SensorPort.S3); //SINISTRA
      final LightSensor lightD = new LightSensor(SensorPort.S2); //DESTRA
      String name = "NXT";
      boolean cond = true;
      LCD.clear();
      LCD.drawString("PRESS TO CREATE CONNECTION", 0, 0);
      LCD.refresh();
      Button.ENTER.waitForPressAndRelease();
      RemoteDevice btrd=null;
      //Fase di connessione
      while (cond) {
            btrd = Bluetooth.getKnownDevice(name);
            if (btrd != null) {
                  LCD.clear();
                  LCD.drawString("DEVICE FOUND", 0, 0);
                  LCD.refresh();
                  cond = false;
            }
      }
      cond = true;
      while(cond) {
            btc = Bluetooth.connect(btrd);
            if (btc != null) {
                  LCD.clear();
                  LCD.drawString("Connection Done", 0, 0);
                  LCD.refresh();
                  cond=false;
            }
      }
```

```
                DataInputStream dis = btc.openDataInputStream();
                DataOutputStream dos = btc.openDataOutputStream();
                pilot.setSpeed(150);
                ForwardBehavior forward = new ForwardBehavior (pilot,lightS,lightD);
                NodoBehavior nodo = new NodoBehavior (clienti, pilot, lightS, lightD, percorso,
                                                percorsoB, dis, dos);
                SearchBehavior search = new SearchBehavior(pilot, lightS,lightD);
                //Wait for ENTER button to be pressed
                Button.ENTER.waitForPressAndRelease();
                Behavior[] bArray = {search, forward, nodo};
                (new Arbitrator(bArray)).start();
        }
}
```

Commenti e note:

- Ovviamente tutti le costanti riguardanti i sensori di luce per i diversi colori devono essere ricavati di volta in volta al cambiare della griglia e dell'ambiente presso cui l'esperimento ha luogo.
- Si noti come l'utilizzo dei behavior renda il codice estremamente leggibile e di facile comprensione.

4.5.3 SearchBehavior.java

La classe SearchBehavior è quella con la minore priorità di esecuzione tra i tre behavior del brick A, e interviene se il robot non segue più la linea nera del percorso: grazie ai due sensori di luce rileva in che verso è necessario sia ruotato il robot e lo riporta nella giusta direzione

```
import lejos.nxt.*;
import lejos.subsumption.*;
import lejos.navigation.*;
import java.util.Vector;
import java.lang.*;

public class SearchBehavior implements Behavior{
        final int blackS    = 440;
        final int blackD    = 370;
        final int minBrownS = 440;
        final int maxBrownS = 600;
        final int orangeS   = 600;
        final int minBrownD = 370;
        final int maxBrownD = 540;
        final int orangeD  = 540;
        private boolean suppress = false;
        public LightSensor lightS;
        public LightSensor lightD;
```

```
        public Pilot pilot;
        public SearchBehavior(Pilot p,LightSensor ls, LightSensor ld){
                super();
                lightS = ls;
                lightD = ld;
                pilot = p;
        }
        public boolean takeControl() {
                return (lightS.readNormalizedValue() < blackS ||
                         lightD.readNormalizedValue() < blackD);
        }
        public void suppress() {
                suppress = true;
                while (suppress) Thread.yield();
        }
        public void action() {
                int sweep = 0;
                if(lightS.readNormalizedValue() < blackS)
                        sweep = 5;
                else
                        if(lightD.readNormalizedValue() < blackD)
                                sweep = -5;
                while (!suppress) {
                        pilot.rotate(sweep,true);
                        while (!suppress && pilot.isMoving())
                                Thread.yield();
                        sweep *= 2;
                }
                pilot.stop();
                suppress = false;
        }
}
```

Commenti e note:

- Come schematizzato in precedenza attraverso la figura 9, basta ricavare quale dei due sensori di luce rileva la linea nera per girare il robot nella direzione corretta in modo, come si può vedere dal codice, decisamente più semplice e immediato di come si sarebbe dovuto fare avendo a disposizione un solo sensore.
- Thread.yield () viene utilizzato per fare in modo che il brick resti libero di fare altre operazioni dato che la rotazione coinvolge solamente i motori.

4.5.4 ForwardBehavior.java

La classe ForwardBehavior richiede il controllo del robot quando esso è correttamente posizionato a cavallo della linea da seguire, situazione che viene effettivamente verificata quando entrambi i sensori, come descritto anche dal metodo takeControl(), rilevano lo sfondo marrone della griglia. Il robot avanza sino a che non incontra un incrocio o sino a perdere la direzione finendo con un sensore sopra la riga nera; in tali casi la classe non richiederà più il controllo del robot in favore degli altri due behavior preposti a risolvere le rispettive situazioni.

```
import lejos.nxt.*;
import lejos.subsumption.*;
import lejos.navigation.*;
import java.util.Vector;
import java.lang.*;

public class ForwardBehavior implements Behavior {
final int blackS    = 440;
        final int blackD    = 370;
        final int minBrownS = 440;
        final int maxBrownS = 600;
        final int orangeS   = 600;
        final int minBrownD = 370;
        final int maxBrownD = 540;
        final int orangeD   = 540;
        public LightSensor lightS;
        public LightSensor lightD;
        public Pilot pilot;
        public ForwardBehavior(Pilot p,LightSensor ls, LightSensor ld){
                super();
                lightS = ls;
                lightD = ld;
                pilot =p;
        }
        public boolean takeControl() {
                return (lightS.readNormalizedValue() >= minBrownS &&
                        lightS.readNormalizedValue() <= maxBrownS &&
                        lightD.readNormalizedValue() >= minBrownD &&
                        lightD.readNormalizedValue() <= maxBrownD);
        }
        public void suppress() {
                pilot.stop();
        }
        public void action() {
                LCD.clearDisplay();
                LCD.drawString("L: " + lightS.readValue(), 0, 1);
```

```
                LCD.drawString("R: " + lightD.readValue(), 0, 2);
                LCD.drawString("Running Behavior: Drive", 0, 3);
                LCD.drawString("Drive", 0, 4);
                pilot.backward();
        }
}
```

4.5.5 NodoBehavior.java

Il behavior NodoBehavior richiede il controllo del robot qualora esso si trovi ad un incrocio: viene infatti verificato che entrambi i sensori di luce rilevino il colore arancione. La classe ha il compito primario di verificare la possibilità che accadano collisioni future, ovvero che ci si trovi in una delle cinque situazioni descritte in precedenza, e di evitarle introducendo preventivamente opportune modifiche al percorso del taxi A. Inoltre ha i compiti di rilevare quando il percorso è terminato (e portare di conseguenza al di fuori della griglia il robot), di gestire i suoni relativi alla presa in carico o al rilascio di un cliente e di aggiornare la direzione del taxi.

```
import lejos.nxt.*;
import lejos.subsumption.*;
import lejos.navigation.*;
import java.util.Vector;
import java.lang.*;
import lejos.nxt.*;
import lejos.nxt.comm.*;
import java.io.*;
import java.util.Random;
import java.lang.Math;
import javax.bluetooth.RemoteDevice;

public class NodoBehavior implements Behavior{
        Pilot pilot;
        LightSensor lightS ;
        LightSensor lightD ;
        Luogo[] percorsoA;
        Luogo[] percorsoB;
        int posizioneB= 0;
        DataInputStream dis ;
        DataOutputStream dos;
        boolean firstTime = true;
        final int blackS = 440;
        final int blackD = 370;
        final int minBrownS = 440;
        final int maxBrownS = 600;
```

```
        final int minBrownD = 370;
        final int maxBrownD = 540;
        final int orangeS = 600;
        final int orangeD = 540;
        int contClienti=1;
        Luogo[] clienti;
        boolean cond= true;
        boolean firstTime2=true;
        char orientamento= 'S';
        public boolean suppress = false;
        public int counter= 0;
        public NodoBehavior(Luogo[] cl,Pilot p,LightSensor ls,LightSensor ld,Luogo[]
movesA,Luogo[] movesB, DataInputStream dis1,DataOutputStream dos1){
                lightS =ls;
                lightD=ld;
                percorsoA = movesA;
                percorsoB = movesB;
                clienti = cl;
                pilot = p;
                dis=dis1;
                dos=dos1;
        }
        public boolean takeControl() {
                return  (lightS.readNormalizedValue() > orangeS ||
                        lightD.readNormalizedValue() > orangeD);
        }
        public void suppress() {
                suppress = true;
                while(suppress) {Thread.yield();}
        }
        public void action() {
                pilot.stop();
                //controllo se il robot è arrivato a fine percorso (ultimo incrocio)
                if (counter==(percorsoA.length-1)){
                        Sound.buzz();
                        Sound.buzz();
                        Sound.buzz();
                        pilot.setSpeed(450);
                        pilot.travel(-50.0f);
                        pilot.rotate(360);
                        System.exit(-1);
                }
                int nextCounter = counter+1;
                Luogo currentA = percorsoA[counter];
                //controllo se ci sono clienti all'incrocio corrente
```

```
if(currentA.uguale(clienti[contClienti])){
        contClienti++;
        Sound.buzz();
        Sound.buzz();
        Sound.buzz();
}
boolean validNextA = true;
boolean validFutureA = true;
Luogo futureA;
Luogo nextA;
if (nextCounter <= (percorsoA.length-1)){
        nextA = percorsoA[nextCounter];
}
else {
        nextA = new Luogo(-1,-1);
        validNextA = false;
}
if (nextCounter <= percorsoA.length-2){
        futureA = percorsoA[nextCounter+1];
}else {
        futureA = new Luogo(-1,-1);
        validFutureA = false;
}
Luogo deviazione1=null;
Luogo deviazione2=null;
//si verifica da che parte girare il robot controllando quale valore cambia
int diffX = currentA.getX()-nextA.getX();
int diffY = currentA.getY()-nextA.getY();
//si chiede la posizione al secondo taxi, se non è a fine percorso
if (firstTime2==true)
        cond=true;
if (posizioneB == percorsoB.length-1){
        while (cond){
                try {
                        LCD.drawString("Closing listener", 0, 0 );
                        LCD.refresh();
                        dos.writeInt(-1);
                        dos.flush();
                        cond = false;
                        firstTime2 = false;
                } catch (IOException ioe) {
                        LCD.drawString("Write Exception", 0, 0);
                        LCD.refresh();
                        cond = true;
```

```
                    }//end catch
                }//end while
            }//end if
            while(cond){
                int temp=0;
                while (cond){
                    try {
                        LCD.drawString("Sending request", 0, temp );
                        temp++;
                        LCD.refresh();
                        dos.writeInt(1);
                        dos.flush();
                        cond = false;
                    } catch (IOException ioe) {
                        LCD.drawString("Write Exception", 0, 0);
                        LCD.refresh();
                        cond = true;
                    }
                }
                cond=true;
                //si ricevono le coordinate del robot2
                while(cond){
                    try {
                        //LCD.clear();
                        //LCD.drawString("Asking position", 0, 0 );
                        //LCD.refresh();
                        posizioneB = dis.readInt();
                        LCD.clear();
                        LCD.drawString("Received:" + posizioneB, 0, 1 );
                        cond=false;
                    } catch (Exception ioe) {
                        LCD.drawString("Read Exception ", 0, 4);
                        LCD.refresh();
                        cond=true;
                    }
                }
                LCD.drawString("Received.. "+posizioneB, 0, 2 );
                LCD.refresh();
                Luogo currentB = percorsoB[posizioneB];
                boolean validNextB = true;
                boolean validFutureB = true;
                Luogo nextB;
                Luogo futureB;
                if (posizioneB <= percorsoB.length-2){
                    nextB = percorsoB[posizioneB+1];
```

```
                    }else {
                            nextB = new Luogo(-1,-1);
                            validNextB = false;
                    }
                    if (posizioneB <= percorsoB.length-3){
                            futureB = percorsoB[posizioneB+2];
                    }else {
                            futureB = new Luogo(-1,-1);
                            validFutureB = false;
                    }
                    //assegnate le posizioni attuali e future, si controllano eventuali collisioni
                    //caso 4
                    if (validNextB && validNextA && currentA.uguale(nextB) &&
nextA.uguale(currentB)){
                            LCD.clear();
                            LCD.drawString("Deviaz:caso4", 0, 0 );
                            LCD.refresh();
                            //si verifica se il brick si muove lungo X oY
                            if( diffX==0) { // varia Y
                                    //controlli per far si che la deviazione punti sempre al centro
                                     mappa
                                    if(currentA.getX()<=2) {
                                    deviazione1=new Luog (currentA.getX()+1,currentA.getY());
                                    deviazione2 = new Luogo(deviazione1.getX(),nextA.getY());
                                    }
                                    else {
                                    deviazione1 =new Luogo(currentA.getX()-1,currentA.getY());
                                    deviazione2 = new Luogo(deviazione1.getX(),nextA.getY());
                                    }
                            }
                            else { //varia X
                                    //controlli per far si che la devizione punti sempre al centro
                                     mappa
                                    if currentA.getY()<=2) {
                                    deviazione1 = new Luogo(currentA.getX(),
                                                            currentA.getY()+1);
                                    deviazione2 = new Luogo(nextA.getX(),deviazione1.getY());
                                    }
                                    else {
                                    deviazione1= new Luogo(currentA.getX(),currentA.getY()-1);
                                    deviazione2 = new Luogo(nextA.getX(),deviazione1.getY());
                                    }
                            }
                    }// fine caso 4
                    //caso 5: A sul percorso di B con rischio di tamponamento
```

```
                    else if (validNextB && validNextA && validFutureB &&
                            currentA.uguale(nextB) && nextA.uguale(futureB)){
                            LCD.clear();
                            LCD.drawString("Deviaz:caso5", 0, 0 );
                            LCD.refresh();
                            if( diffX==0) {
                                    if(currentA.getX()<=2) {
                                    deviazione1 = new Luogo(currentA.getX()+1,
                                                            currentA.getY() );
                                    deviazione2 = new Luogo(deviazione1.getX(),nextA.getY());
                                    }
                                    else {
                                    deviazione1 = new Luogo(currentA.getX()-1,
                                                            currentA.getY());
                                    deviazione2 = new Luogo(deviazione1.getX(),nextA.getY());
                                    }
                            }
                            else {
                                    if(currentA.getY()<=2) {
                                    deviazione1 = new Luogo(currentA.getX(),
                                                             currentA.getY()+1);
                                    deviazione2 = new Luogo(nextA.getX(),deviazione1.getY());
                                    }
                                    else {
                                    deviazione1 =new Luogo(currentA.getX(),currentA.getY()-1);
                                    deviazione2 = new Luogo(nextA.getX(),deviazione1.getY());
                                    }
                            }
                    }//fine caso 5
                    //caso 2
                    else if(validNextB && validNextA && validFutureB &&
                            currentA.uguale(futureB) && nextA.uguale(nextB)){
                            LCD.clear();
                            LCD.drawString("Deviaz:caso2", 0, 0 );
                            LCD.refresh();
                            if( diffX==0) {
                                    if(currentA.getX()<=2) {
                                    deviazione1 = new Luogo(currentA.getX()+1,
                                                             currentA.getY());
                                    deviazione2 = new Luogo(deviazione1.getX(),nextA.getY());
                                    }
                                    else {
                                    deviazione1= new Luogo(currentA.getX()-1,currentA.getY());
                                    deviazione2 = new Luogo(deviazione1.getX(),nextA.getY());
                                    }
```

```
                    }
                    else {
                        if(currentA.getY()<=2) {
                        deviazione1 = new Luogo(currentA.getX(),
                                        currentA.getY()+1);
                        deviazione2 = new Luogo(nextA.getX(),deviazione1.getY());
                        }
                        else {
                        deviazione1 =new Luogo(currentA.getX(),currentA.getY()-1);
                        deviazione2 = new Luogo(nextA.getX(),deviazione1.getY());
                        }
                    }
                }//fine caso 2
                //caso 3
                else if(validNextB && validNextA && validFutureB && validFutureA
                         && nextA.uguale(futureB) && futureA.uguale(nextB)){
                    LCD.clear();
                    LCD.drawString("Deviaz:caso3", 0, 0 );
                    LCD.refresh();
                    if( diffX==0) {
                        if(currentA.getX()<=2) {
                        deviazione1 = new Luogo(currentA.getX()+1,
                                        currentA.getY());
                        deviazione2 = new Luogo(deviazione1.getX(),nextA.getY());
                        }
                        else {
                        deviazione1 =new Luogo(currentA.getX()-1,currentA.getY());
                        deviazione2 = new Luogo(deviazione1.getX(),nextA.getY());
                        }
                    }
                    else {
                        if(currentA.getY()<=2) {
                        deviazione1 = new Luogo(currentA.getX(),
                                        currentA.getY()+1);
                        deviazione2 = new Luogo(nextA.getX(),deviazione1.getY());
                        }
                        else {
                        deviazione1 =new Luogo(currentA.getX(),currentA.getY()-1);
                        deviazione2 = new Luogo(nextA.getX(),deviazione1.getY());
                        }
                    }
                    LCD.drawString(deviazione1.toString(), 0, 1 );
                    LCD.drawString(deviazione2.toString(), 0, 2 );
                    LCD.refresh();
                //caso 1: semplice incrocio di traiettorie, il robot A attende il passaggio di B
```

```
                else if(validNextB && validNextA && nextA.uguale(nextB)){
                        LCD.clear();
                        LCD.drawString("Deviaz:caso1", 0, 0 );
                        LCD.refresh();
                        deviazione1 = null;
                        deviazione2 = null;
                        try{
                                Thread.sleep(7000);
                        }catch (Exception e){};
                        cond=true;
                } //fine caso 1
                else {
                        LCD.clear();
                        LCD.drawString("Nessuna deviaz", 0, 0 );
                        LCD.refresh();
                        cond=false;
                }
        }
        //si verifica se ci sono da effettuare modifiche al percorso
        if( deviazione1 != null && deviazione2 != null && counter>=2){
                //aggiornamento dell'array del percorso con inserimento della deviazione
                percorsoA[counter-2] = currentA;
                percorsoA[counter-1] = deviazione1;
                percorsoA[counter] = deviazione2;
                counter= counter-2;
                Sound.playTone(1000,200);
                LCD.clear();
                LCD.drawString("AGG ARRAY", 0, 0 );
                LCD.refresh();
        }
        diffX = percorsoA[counter].getX()-percorsoA[counter+1].getX();
        diffY = percorsoA[counter].getY()-percorsoA[counter+1].getY();
        //aggiornamento dell'orientamento del robot
        switch(diffX){
        case +1: switch(orientamento){
                        //si deve andare in direzione nord
                        case 'N': pilot.travel(-12.0f);
                                break;
                        case 'E': orientamento = 'N';
                                pilot.travel(-9.0f);
                                pilot.rotate(90); //sinistra
                                break;
                        case 'S': orientamento = 'N';
                                pilot.rotate(180);
                                break;
```

```
                        case 'O': orientamento = 'N';
                                pilot.travel(-9.0f);
                                pilot.rotate(-90); //destra
                                break;
                        default:break;
                }
        break;
        case -1: // si deve andare in direzione sud
                switch(orientamento){
                        case 'S': pilot.travel(-12.0f);
                                break;
                        case 'O': orientamento = 'S';
                                pilot.travel(-9.0f);
                                pilot.rotate(90);
                                break;
                        case 'N': orientamento = 'S';
                                pilot.rotate(180);
                                break;
                        case 'E': orientamento = 'S';
                                pilot.travel(-9.0f);
                                pilot.rotate(-90);
                                break;
                        default:break;
                }
        break;
        default:break;
        }
        switch(diffY){
        case +1: // si deve andare in direzione ovest
                switch(orientamento){

                        case 'O': pilot.travel(-12.0f);
                                break;
                        case 'N': orientamento = 'O';
                                pilot.travel(-9.0f);
                                pilot.rotate(90); //sinistra
                                break;
                        case 'E': orientamento = 'O';
                                pilot.rotate(180);
                                break;
                        case 'S': orientamento = 'O';
                                pilot.travel(-9.0f);
                                pilot.rotate(-90); //destra
                                break;
                        default:break;
```

```
                }
            break;
            case -1: // si deve andare in direzione est
                switch(orientamento){
                    case 'E': pilot.travel(-12.0f);
                        break;
                    case 'S': orientamento = 'E';
                        pilot.travel(-9.0f);
                        pilot.rotate(90); //sinistra
                        break;
                    case 'O': orientamento = 'E';
                        pilot.rotate(180);
                        break;
                    case 'N': orientamento = 'E';
                        pilot.travel(-9.0f);
                        pilot.rotate(-90); //destra
                        break;
                    default:break;
                }
            break;
            default: break;
            }
            counter++;
            suppress = false;
        }
}
```

Commenti e note:

- Data la lunghezza della classe in questione, si è preferito inserire nel codice delle brevi spiegazioni riguardo il compito di ciascuna parte.

4.5.6 Classi in comune tra i due taxi: Luogo e Importer

Luogo è una classe che, nell'ambito della logica di programmazione ad oggetti, rappresenta gli incroci della griglia utilizzata dai brick; Importer, invece, è una classe di supporto per la creazione di array di luoghi a partire da una stringa di testo. La stringa in questione non viene ottenuta accedendo direttamente ai file di testo: quest'ultima funzione è eseguita dalla procedura main, presente rispettivamente in RobotA.java e RobotB.java, che successivamente invoca Importer fornendo la riga di testo come parametro.

```
public class Luogo{
	int x;
	int y;
	public Luogo(int a, int b){
		x=a;
		y=b;
	}
	public int getX(){
		return x;
	}
	public void setX(int a){
		x=a;
	}
	public int getY(){
		return y;
	}
	public void setY(int a){
		y=a;
	}
	public boolean uguale(Luogo a){
		if(x==a.getX() && y==a.getY())
			return true;
		else
			return false;
	}
	public String toString() {
		return "LUOGO> X:"+x+" Y:"+y;
	}
}
```

```
import java.lang.Integer;
import lejos.nxt.*;
import lejos.subsumption.*;
import lejos.navigation.*;
import java.util.Vector;
import java.lang.*;
import lejos.nxt.*;
import lejos.nxt.comm.*;
import java.io.*;
import java.util.Random;
import java.lang.Math;
import javax.bluetooth.RemoteDevice;

public class Importer {
```

```
	public Importer() {
		super();
	}
	public Luogo[] importa(String txt) {
		String pathTxt = txt;
		Luogo placesArray[] = new Luogo[50];
		String path=txt;
		String prologString = path;
		int inizio = prologString.indexOf('[') + 1;
		int fine = prologString.indexOf(']');
		prologString = prologString.substring(inizio,fine);
		int contaPosizioni = 0;
		int j=0;
		int punt=0;
		String p = "";
		for(int i=0;i<prologString.length();i++) {
			if (prologString.charAt(i)==',') {
				p = prologString.substring(j,i);
				int a = Integer.parseInt(p.substring(0,1));
				int b = Integer.parseInt(p.substring(1,2));
				Luogo temp = new Luogo(a,b);
				placesArray[punt]=temp;
				j=i+1;
				punt++;
			}
		}
		p = prologString.substring(j);
		int a = Integer.parseInt(p.substring(0,1));
		int b = Integer.parseInt(p.substring(1,2));
		Luogo temp = new Luogo(a,b);
		placesArray[punt]=temp;
		Luogo placesArray2[] = new Luogo[punt+1];
		for(int i=0;i<=punt;i++) {
			placesArray2[i] = placesArray[i];
		}
		return placesArray2;
	}
}
```

4.5.7 RobotB.java

La classe oggetto di questo paragrafo contiene la procedura main del robot B: essa gestisce i file di testo e stabilisce la connessione Bluetooth con il primo taxi; inoltre al suo interno definisce due behavior analoghi a ForwardBehavior e SearchBehavior del brick A.

```
import lejos.nxt.*;
import lejos.subsumption.*;
import lejos.navigation.*;
import java.util.Vector;
import java.lang.*;
import lejos.nxt.comm.*;
import java.io.*;
import java.util.Random;
import java.lang.Math;
import javax.bluetooth.RemoteDevice;

public class RobotB {
	public static void main (String[] aArg) throws Exception{
		Importer imp = new Importer();
		File data = new File("pathB.txt");
		String strB ="";
		try {
			InputStream is = new FileInputStream(data);
			DataInputStream din = new DataInputStream(is);
			while (is.available() > 0) {
				strB = strB + (char)din.read();
			}
			din.close();
		} catch (IOException ioe) {
			LCD.drawString("read exception", 0, 0);
		}
		data = new File("cpathB.txt");
		String strC ="";
		try {
			InputStream isC = new FileInputStream(data);
			DataInputStream dinC = new DataInputStream(isC);
			while (isC.available() > 0) {
				strC = strC + (char)dinC.read();
			}
			dinC.close();
		} catch (IOException ioe) {
			LCD.drawString("read exception", 0, 0);
		}
		Luogo[] percorso = imp.importa(strB);
```

```
Luogo[] clienti = imp.importa(strC);
Dati dati = new Dati(0);
final Pilot pilot = new Pilot(5.6f,12.5f,Motor.A, Motor.C, true);
final LightSensor lightS = new LightSensor(SensorPort.S3);
final LightSensor lightD = new LightSensor(SensorPort.S2);
pilot.setSpeed(150);
final int blackS = 350;
final int blackD = 450;
final int minBrownS = 350;
final int maxBrownS = 470;
final int orangeS = 470;
final int minBrownD = 450;
final int maxBrownD = 580;
final int orangeD = 580;
boolean cond= true;
LCD.clearDisplay();
LCD.drawString("Press to start listener", 0, 1);
LCD.refresh();
Button.ENTER.waitForPressAndRelease();
BTConnection btc = null;
while (cond){
        btc = Bluetooth.waitForConnection();
        if(btc!=null)
                cond = false;
}
DataInputStream dis = btc.openDataInputStream();
DataOutputStream dos = btc.openDataOutputStream();
//avvio del thread che ascolterà lo stream in ingresso
ListenerThread list = new ListenerThread(dis,dos,dati);
list.start();
Behavior DriveForward = new Behavior(){
        public boolean takeControl() {
                return
                (lightS.readNormalizedValue() >= minBrownS &&
                 lightS.readNormalizedValue() <= maxBrownS &&
                 lightD.readNormalizedValue() >= minBrownD &&
                 lightD.readNormalizedValue() <= maxBrownD
                );
        }
        public void suppress() {
                pilot.stop();
        }
        public void action() {
                LCD.clearDisplay();
```

```
                        LCD.drawString("L: " + lightS.readValue(), 0, 1);
                        LCD.drawString("R: " + lightD.readValue(), 0, 2);
                        LCD.drawString("Running Behavior: Drive", 0, 3);
                        LCD.drawString("Drive", 0, 4);
                        pilot.backward();
                }
        };
        Behavior OffLine = new Behavior(){
                private boolean suppress = false;
                public boolean takeControl() {
                        return (lightS.readNormalizedValue() < blackS ||
                                lightD.readNormalizedValue() < blackD);
                }
                public void suppress() {
                        suppress = true;
                        while (suppress) Thread.yield();
                }
                public void action() {
                        int sweep = 0;
                        if(lightS.readNormalizedValue() < blackS)
                                sweep = 5;
                        else if(lightD.readNormalizedValue() < blackD)
                                sweep = -5;
                        while (!suppress) {
                                pilot.rotate(sweep,true);
                                while (!suppress && pilot.isMoving()) Thread.yield();
                                sweep *= 2;
                        }
                        pilot.stop();
                        suppress = false;
                }
        };
        NodoBehavior nodo = new NodoBehavior(clienti,dati,pilot,lightS,
                                        lightD,percorso, list);
        Button.ENTER.waitForPressAndRelease();
        Behavior[] bArray = {OffLine, DriveForward, nodo};
        (new Arbitrator(bArray)).start();
    }
}
```

Commenti e note:

- Il codice della classe e dei due behavior al suo interno è analogo a quanto visto per il robot A; le differenze con quest'ultimo sono la gestione della comunicazione della posizione, gestita da ListenerThread che si mette in ascolto di eventuali richieste da parte del taxi A, e l'accesso sincronizzato da parte dei thread alla posizione corrente del brick B, garantita dalla classe Dati.

4.5.8 NodoBehavior.java (per taxi B)

La classe suddetta si differenzia da quanto visto per il brick A sostanzialmente per l'assenza della gestione delle collisioni in quanto il robot B comunica solamente la propria posizione. Rimangono pressoché invariate la gestione dell'orientamento e dei movimenti del robot, il controllo riguardante la presenza o meno di clienti a un incrocio e la verifica della situazione di fine percorso.

```
import lejos.nxt.*;
import lejos.subsumption.*;
import lejos.navigation.*;
import java.util.Vector;
import java.lang.*;

public class NodoBehavior implements Behavior{
        Dati dati;
        Pilot pilot;
        LightSensor lightS ;
        LightSensor lightD ;
        Luogo[] percorso;
        Luogo[] clienti;
        ListenerThread t;
        char orientamento = 'S';
        final int blackS = 350;
        final int blackD = 450;
        final int minBrownS = 350;
        final int maxBrownS = 470;
        final int orangeS = 470;
        final int minBrownD = 450;
        final int maxBrownD = 580;
        final int orangeD = 580;
        int contClienti = 1;
        public boolean suppress = false;
        public NodoBehavior(Luogo[] cl,Dati po,Pilot p,LightSensor ls,LightSensor ld,
                        Luogo[] moves, ListenerThread lt){
                clienti = cl;
                dati = po;
                lightS =ls;
                lightD=ld;
                percorso = moves;
                pilot = p;
                t = lt;
        }
```

```
public boolean takeControl() {
        return (lightS.readNormalizedValue() > orangeS ||
                lightD.readNormalizedValue() > orangeD);
}
public void suppress() {
        suppress = true;
        while(suppress) {Thread.yield();}
}
public void action() {
        pilot.stop();
        pilot.setSpeed(150);
        int pos = dati.getInt();
        //fine percorso?
        if (pos==percorso.length-1){
                Sound.buzz();
                Sound.buzz();
                Sound.buzz();
                pilot.setSpeed(450);
                pilot.travel(-50.0f);
                pilot.rotate(360);
                try{
                        Thread.sleep(10000);
                        t.join();
                        System.exit(0);
                }
                catch(Exception e){};
        }
        Luogo currentA = percorso[pos];
        Luogo nextA = percorso[pos+1];
        //controllo presenza o rilascio clienti
        if(currentA.uguale(clienti[contClienti])){
                contClienti++;
                Sound.buzz();
                Sound.buzz();
                Sound.buzz();
        }
//si rileva la parte dalla quale girare verificando su quale asse il brick deve muoversi
        int diffX = currentA.getX()-nextA.getX();
        int diffY = currentA.getY()-nextA.getY();
        //aggiornamento direzione
        switch(diffX){
                case +1: //verso nord
                        switch(orientamento){
                                case 'N': pilot.travel(-12.0f);
                                        break;
```

```
                        case 'E': orientamento = 'N';
                                pilot.travel(-9.0f);
                                pilot.rotate(90); //sinistra
                                break;
                        case 'S': orientamento = 'N';
                                pilot.rotate(180);
                                break;
                        case 'O': orientamento = 'N';
                                pilot.travel(-9.0f);
                                pilot.rotate(-90); //destra
                                break;
                        default:break;
                }
        break;
        case -1: //verso sud
                switch(orientamento){
                        case 'S': pilot.travel(-12.0f);
                                break;
                        case 'O': orientamento = 'S';
                                pilot.travel(-9.0f);
                                pilot.rotate(90);
                                break;
                        case 'N': orientamento = 'S';
                                pilot.rotate(180);
                                break;
                        case 'E': orientamento = 'S';
                                pilot.travel(-9.0f);
                                pilot.rotate(-90);
                                break;
                        default:break;
                }
        break;
        default:break;
}
switch(diffY){
        case +1: //verso ovest
                switch(orientamento){
                        case 'O': pilot.travel(-12.0f);
                                break;
                        case 'N': orientamento = 'O';
                                pilot.travel(-9.0f);
                                pilot.rotate(90); //sinistra
                                break;
                        case 'E': orientamento = 'O';
```

```
                                        pilot.rotate(180);
                                        break;
                                case 'S': orientamento = 'O';
                                        pilot.travel(-9.0f);
                                        pilot.rotate(-90); //destra
                                        break;
                                default:break;
                        }
                break;
                case -1: //verso est
                        switch(orientamento){
                                case 'E': pilot.travel(-12.0f);
                                        break;
                                case 'S': orientamento = 'E';
                                        pilot.travel(-9.0f);
                                        pilot.rotate(90); //sinistra
                                        break;
                                case 'O': orientamento = 'E';
                                        pilot.rotate(180);
                                        break;
                                case 'N': orientamento = 'E';
                                        pilot.travel(-9.0f);
                                        pilot.rotate(-90); //destra
                                        break;
                                default:break;
                        }
                        break;
                        default: break;
                }
                if(dati.getInt()<percorso.length-1) {
                        dati.setInt(pos+1);
                }
        }// end action()
}
```

4.5.9 ListenerThread e la classe Dati

ListenerThread è un thread che si occupa di ascoltare il canale di comunicazione Bluetooth: quando viene inviata una richiesta dal robot A, la classe la analizza ed agisce di conseguenza inviando i dati richiesti o chiudendo il thread, a seconda che sia una domanda riguardante la posizione corrente del brick B o un avviso di fine connessione.

```
import lejos.nxt.*;
import lejos.nxt.comm.*;
import java.io.*;
import java.lang.Math;

public class ListenerThread extends Thread {
	public DataInputStream dis;
	public DataOutputStream dos;
	Dati dati;
	public ListenerThread(DataInputStream input, DataOutputStream output,Dati d) {
		dis = input;
		dos = output;
		dati = d;
	}
	public void run(){
		int n = 0;
		boolean valido = true;
		while (valido){
			boolean cond= true;
			try{
				n = dis.readInt();
			}catch(Exception e){cond = false;}
			LCD.drawString("Ricevo richiesta", 0, 3 );
			LCD.refresh();
			while (cond){
				//condizione di chiusura del thread
				if (n==-1){
					valido = false;
					cond= false;
				}
				if (n==1) {
					try{
						dos.writeInt(dati.getInt());
						dos.flush();
						cond=false;
						LCD.drawString("Rispondo.. "+dati.getInt(), 0, 4 );
						LCD.refresh();
					}catch(Exception e){};
				}
			}
		}
	}
}
```

La classe Dati garantisce un accesso sincrono da parte di ListenerThread e NodoBehavior ai propri metodi e in particolare alla variabile intera x, che è l'indice dell'array di luoghi corrispondente alla posizione corrente del brick B.

```
public class Dati {
        public int x;
        public Dati (int a) {
                x=a;
        }
        public synchronized int getInt(){
                return x;
        }
        public synchronized void setInt(int a){
                 x = a;
        }
}
```

Capitolo 5

Conclusioni

Il firmware sostituivo LeJOS aggiunge indubbiamente ulteriore flessibilità ai brick LEGO. Le limitazioni introdotte per far sì che la Virtual Machine potesse funzionare correttamente senza impiegare troppe risorse nel brick LEGO non si sono tradotte in forti restrizioni nella programmazione del robot da parte dell'utente, che può controllare sostanzialmente l'intero hardware senza particolari vincoli. Si segnalano, in presenza di alcuni metodi, la necessità di inserire dei ritardi affinché essi funzionino correttamente, o comunque, in altre occasioni, delle soluzioni non proprio ottimali ma necessarie per superare alcuni problemi: l'auspicio è che questi inconvenienti possano essere risolti, come del resto è stato già fatto in precedenza per altri bug, con le successive release di LeJOS. Nonostante questo, comunque, LeJOS risulta affidabile e di gran lunga tra i migliori firmware e ambienti operativi per brick LEGO; inoltre presenta l'indubbio vantaggio di avvicinare il vasto pubblico abituato alla programmazione Java alla robotica senza lo sforzo di dover imparare altri linguaggi specifici.

In Internet sono presenti numerosi siti, ma anche semplici blog, che trattano in particolare i brick, il loro hardware e il firmware LeJOS. Purtroppo gli algoritmi e i programmi che si possono trovare non sono altrettanti: è abbastanza difficile reperire del codice significativo o che comunque possa essere riutilizzato in altri problemi; solitamente ci si limita a mostrare tramite video i risultati pratici del proprio lavoro e a inserire spezzoni dei programmi sviluppati solamente per ricevere aiuto nel risolvere eventuali errori.

Bibliografia

[1] B. Bagnall, *Core Lego Mindstorms programming: unleash the power of the Java platform*, Prentice Hall, 2002.
[2] J. A. Breña Moral, *Develop LeJOS programs step by step*, <http://www.juanantonio.info/lejos-ebook/>, 2008, agg. 2009.

Sitografia

http://lejos.sourceforge.net
http://mindstorms.lego.com
http://www.adrirobot.it
http://www.philohome.com
http://mercurio.srv.dsi.unimi.it/~ornaghi/IntelligenzaArtificiale
http://weblogs.java.net/blog/2008/02/02/controlling-lego-mindstorms-java-lejos
http://digilander.libero.it/Valter_NXT
http://blog.electricbricks.com
http://hansf10.blogspot.com