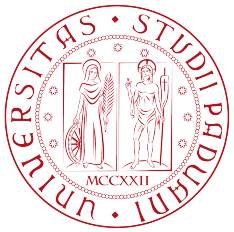

# UNIVERSITÀ DEGLI STUDI DI PADOVA

**FACOLTÀ DI INGEGNERIA**
**Corso di Laurea in Ing. Informatica**

**Tesi di laurea**

# SIMULAZIONE VITA ANIMALE CON LEGO NXT Bluetooth e NXTCam

**Relatore:**
**Prof.Michele Moro**

**Laureando:**
**Zahir Fetovski**

Anno Accademico 2010/2011

# Indice

# Introduzione

Il progetto, realizzato durante l'anno accademico 2009/2010, prevedeva la costruzione di tre robot e l'implementazione del relativo software, sfruttando l'hardware dei kit Lego Mindstorm, al fine di riprodurre una simulazione di vita animale da esporre presso il museo civico di Rovereto in occasione dell'evento Discovery on Film 2010, intitolato : Ispirati dalla Natura.

## Argomenti approfonditi

L'intero lavoro è stato realizzato in collaborazione con il collega Franco Marangoni, con degli approfondimenti diversificati su alcune specifiche di funzionamento e conseguenti considerazioni. In particolare in questa tesi viene approfondito il protocollo bluetooth utilizzato per la connessione tra pi ù robot, e il delicato utilizzo della NXTCAM , la videocamera montata su uno dei tre robottini.

## Discovery On film : Progetto del museo civico di Rovereto

Per avvicinare scienza e tecnologia, favorendo l'essenziale interazione tra ricerca, didattica ed industria, è nata nel 2001 la Mostra di documentari Discovery on film. Una rassegna che vede per due giornate Rovereto e il suo Museo protagonisti del dibattito scientifico internazionale, grazie alla proiezione di audiovisivi provenienti dal vasto repertorio del Prix Leonardo (il maggiore festival del film scientifico, in programma annualmente a Parma), ma grazie anche ad incontri con grandi personalità della ricerca mondiale. Tra questi, nell'edizione 2002, il padre della robotica Antal Bejczy.

# Capitolo 1

# Robot esposti

Di seguito una breve descrizione del ruolo svolto da ciascuno dei tre robot realizzati, con un breve accenno ai dettagli tecnici, trattati in maniera più approfondita nella tesi del collega F. Marangoni.

## 1.1 Beezer


Figura 1.1: Beezer

Beezer rappresenta un insetto che ha come obiettivo quello di raggiungere un ipotetico fiore percorrendo il tragitto più breve ed evitando gli eventuali ostacoli sul percorso. L'intero percoso è situato su una griglia composta da quadrati di 40 cm x 40 cm, all'interno dei quali possono essere posti degli ostacoli o l'obiettivo stesso. Arrivato a destinazione, vi può trovare il goal oppure un oggetto diverso: nel primo caso trasmette il percorso all'altro robot, Molly mentre nel secondo, prosegue la ricerca nell'intorno fino a che non giunge a destinazione.

### 1.1.1 Dettagli tecnici

La ricerca del cammino minimo viene effettuata su di una mappa precaricata nel brick, corrispondende alla griglia su cui si muove lo stesso robot, utilizzando il wavefront algorithm. L'identificazione del goal avviene tramite il riconoscimento del colore con l'NXTCam. La trasmissione del percorso al robot Molly, avviene tramite il bluetooth integrato nel brick, che vedremo più in dettaglio nei capitoli successivi. Ad ogni step sulla griglia dove si svolge l'esperienza, viene effettuato il riallineamento, indispensabile per far muovere il robot correttamente.

## 1.2 Molly


Figura 1.2: Molly

A Molly non è stata implementa la funzionalità di ricerca fiore come a Beezer, perciò attende che quest'ultimo le invii il percorso una volta giunto a destinazione, in modo tale da poterlo ripercorrere passo-passo per arrivare a destinazione.

### 1.2.1 Dettagli tecnici

Molly dunque riceve da Bezeer, grazie alla trasmissione via bluetooth, un vettore contenente la sequenza di step da eseguire per raggiungere lo stesso goal di Bezeer. Non necessita di riallineamento perchè grazie alla dimensione predefinita della griglia, ai cingoli, e al percorso predefinito ricevuto, risulta non necessario per lo spostamento corretto all'interno della griglia.

## 1.3 Spike

Anche Spike si muove all'interno della griglia, con percorso casuale ma evitando gli eventuali ostacoli. Inoltre gli è stata implementata la funzionalità di attaccare con il pungiglione un ipotetico nemico quando si sente minacciato.

### 1.3.1 Dettagli tecnici

Rilevato l'imprevisto, Spike si ferma, se la distanza è maggiore di un valore predefinito lo aggira, altrimenti esegue un' ulteriore misura e se questa è minore della precedente rileva il pericolo e attacca con il pungiglione. La seconda lettura deve essere immediata al fine di simulare il riflesso nel modo più reale possibile. Gli spostamenti avvengono utilizzando una variabile che assumendo valori random ad intervalli regolari stabilisce la direzione che Spike dovrà intraprendere.


Figura 1.3: Spike

# Capitolo 2

# Linguaggio NXC

## 2.1 BrixCC

Il software utilizzato per la programmazione dei Robot è BrixCC . BrixCC è un ambiente di programmazione che permette la scrittura dei programmi, la loro compilazione, la ricerca degli errori, ed il trasferimento del codice al robot. Può gestire programmi scritti in vari linguaggi, ma quello che è stato utilizzato per l'implementazione dei robot è NXC ( Not Exactly C), molto simile al C. BrixCC è un programma open source, liberamente scaricabile ed utilizzabile, e gira su sistemi operativi Microsoft.

## 2.2 Avvio BrixCC

All'avvio del Brixcc il programma verifica se è attivo il collegamento al robot. Compare uno screenshot che richiede alcune impostazioni: quale tipo di interfaccia utilizzare, il tipo di robot che si intende programmare ed il tipo di firmware caricato sul robot.

## 2.3 Compilazione

Il menù compile,è il cuore del programma.

**Compile-Compile -** Compila il programma appena scritto. Se trova degli errori, il compilatore li segnala con dei messaggi espliciti (che compaiono in basso), indispensabili per capire in che punto del programma c'è l'errore e la natura dell' errore stesso.

**Compile-Download -** Compila il programma e lo trasferisce instantaneamente all' NXT. A questo punto basta premere sull' NXT il tasto di avvio del programma ed il robot eseguirà le istruzioni caricate.

**Compile-Download and Run -** Compila il programma, lo trasferisce all'NXT, e lo esegue immediatamente.

## 2.4 Codice NXC

Ogni programma scritto in NXC deve contenere un task ed in particolare deve contenere il task main. I task possono essere anche più di uno, ma qualora altri task esistano, vengono chiamati dal task principale. Ogni task inizia con una parentesi graffa aperta e termina con una graffa chiusa, per indicare rispettivamente l'inizio e la fine delle istruzioni che ne fanno parte. Contenute tra le parentesi, ci sono le istruzioni semplici, gli statement.

### 2.4.1 Esempio istruzioni NXC

```
OnFwd(OUT_A, 75);
```

Questa istruzione permette al robot di muovere un motore in avanti OnFwd sta per Accendi=On e muovi in avanti = Forward = Fwd. Un modulo NXT può gestire fino a 4 motori. In questo esempio a muoversi sarà quello collegato all uscita A, al 75% della velocità massima. L'istruzione poi termina con il classico punto e virgola. Le 4 porte di uscita tramite cui il modulo NXT gestisce i motori sono identificate da una lettera maiuscola: A, B, C, D)

```
Wait (4000);
```

Attendi 4 secondi. Se le due istruzioni appena descritte vengono eseguite in successione indicherebbero all'NXT di muoversi in avanti per 4 secondi esatti. Infatti l'argomento (4000) impone al processore di mantenere attive le uscite per 4000 millesimi di secondo.

```
OnRev (OUT_AC, 75);
```

Questa istruzione impone al programma di muovere due motori indietro. OnRev sta per Accendi=On e muovi indietro = Reverse = Rev. OUTAC Specifica di attivare contemporaneamente le uscite A e C.

## 2.5 Task e Subroutine

I programmi NXC possono essere costituiti da molteplici task. L'uso di molteplici task e di subroutine consente di rendere i programmi più leggibili e più compatti. Un programma NXC consiste di un numero di task a piacere, tra 1 e 255. Il task main deve sempre esistere,dato che è il principale task del programma, ed il primo ad essere eseguito. Gli altri task vengono eseguiti qualora richiamati da un altro task in esecuzione.

### 2.5.1 Variabili Mutex e Programmazione concorrente

È importante comprendere che i task avviati vengono eseguiti in concorrenza. Questo comporta il dover gestire il problema del tentativo simultaneo di accedere alla medesima risorsa da parte dei task. Se queste molteplici richieste effettuate da parte di ogni task non vengono gestite, si va incontro a risultati inattesi. Ad esempio,se entrambi i task cercano di far muovere i motori simultaneamente, chi dei due (o più) task attivi ha la precedenza?. Per ovviare a questo tipo di problema, vengono utilizzate le variabili mutex (mutual exclusion). Possiamo assegnare queste variabili alle funzioni Acquire e Release e racchiudere le parti critiche di codice tra di esse. In questo modo saremo certi che il controllo (ad esempio dei motori) verrà acquisito da un solo task per volta.

### 2.5.2 Uso dei semafori

Esiste però un metodo alternativo all' uso delle variabili mutex. Una tecnica standard per risolvere il problema è quello di usare una variabile per indicare quale dei task sta usando i motori. Gli altri task non sono abilitati ad usare i motori finch é il primo task non indica, usando la variabile, che ha liberato le risorse. Una variabile di questo tipo è chiamata semaforo. Facciamo un esempio con sem variabile semaforo (equiparabile ad una variabile mutex). Poniamo che un valore uguale a 0 indica che nessun task sta usando i motori (la risorsa è libera).

```
until (sem == 0);
sem = 1; //Acquire(sem);
// Do something with the motors
// critical region
sem = 0;
//Release(sem);
```

Ora, qualunque task voglia usare i motori eseguirà i seguenti comandi:per prima cosa bisogna che nessuno utilizzi la risorsa, ciò significa attendere che il valore della variabile sia 0. Poi, ne prendiamo il controllo ponendo la variabile sem uguale ad 1.Ora possiamo controllare i motori. Alla fine, quando il task non ha più bisogno di utilizzare la risorsa,risetta la variabile sem al valore di zero, segnalando cos che la risorsa è stata liberata alle altre sezioni del programma. [5] [6]

# Capitolo 3

# Protocollo Bluetooth

Figura 3.1: Logo ufficiale Bluetooth

## 3.1 Informazioni generali

Lo standard Bluetooth, è stato progettato con l'obiettivo primario di ottenere bassi consumi, con conseguente raggio di azione ridotto (da 1 a 100 metri) e un basso costo di produzione per i dispositivi compatibili. Lo standard doveva consentire il collegamento wireless tra periferiche come stampanti, tastiere, telefoni, microfoni, ecc. a computer o PDA. Attualmente più di un miliardo di dispositivi montano un'interfaccia Bluetooth. La tecnologia Bluetooth opera nella banda di frequenze tra 2,4 e 2,5 GHz, che è denominata ISM (Industrial Scientific Medical) ed è libera da ogni licenza di utilizzo. Bluetooth usa una tecnica trasmissiva di tipo FHSS (Frequency Hopping Spread Spectrum a spettro diffuso con salti di frequenza), in cui la frequenza non è fissata ma varia tra 79 frequenze diverse (hopping) ad intervalli regolari, secondo una sequenza pseudocasuale. Ciò consente di contrastare meglio l'interferenza. I dati possono essere trasmessi su canali

sincroni (SCO, Synchronous Connection-Oriented) o asincroni (ACL, Asynchronous Connection- Less). I primi sono tipicamente usati per il traffico audio, in cui è necessario mantenere la coerenza temporale, mentre i secondi sono utilizzati per i dati. Ogni dispositivo Bluetooth è in grado di gestire simultaneamente la comunicazione con altri 7 dispositivi, sebbene essendo un collegamento di tipo master-slave solo un dispositivo per volta può comunicare con il server. Questa rete minimale viene chiamata piconet. Le specifiche Bluetooth consentono di collegare due piconet in modo da espandere la rete.

La versione 1.1 e 1.2 del Bluetooth gestisce velocità di trasferimento fino a 723,1 kb/s. La versione 2.0 gestisce una modalità ad alta velocità che consente comunicazioni fino a 3 Mb/s. Questa modalità però aumenta la potenza assorbita. La nuova versione utilizza segnali più brevi e quindi riesce a dimezzare la potenza richiesta rispetto al Bluetooth 1.2 (a parità di traffico inviato).

## 3.2 Classi dei dispositivi

I dispositivi Bluetooth si dividono in 3 classi fondamentali :

| Classe | Potenza (mW) | Potenza (dBm) | Distanza (m) |
|---|---|---|---|
| Classe 1 | 100 | 20 | ~ 100 |
| Classe 2 | 2,5 | 4 | ~ 10 |
| Classe 3 | 1 | 0 | ~ 1 |

Figura 3.2: Classi dei dispositivi BT

## 3.3 Caratteristiche tecniche

### 3.3.1 Temporizzazione e clock

La tecnologia Bluetooth prevede di sincronizzare la maggior parte delle operazioni con un clock in tempo reale. Il clock Bluetooth è realizzato con un

contatore a 28 bit che viene posto a 0 all'accensione del dispositivo e subito dopo continua senza mai fermarsi, incrementandosi ogni 312,5 $\mu$ s. Il ciclo del contatore copre approssimativamente la durata di un giorno (312,5 $\mu$ 228 23,3 ore). Ogni dispositivo Bluetooth ha il suo native clock (CLKN) che controlla la temporizzazione. Oltre a questo valore, proprio di ogni dispositivo, Bluetooth definisce altri due clock:

-**CLK**: Questo è il clock della piconet, coincide con il CLKN dell'unità master della piconet. Tutte le unità attive nella piconet devono sincronizzare il proprio CLKN con il CLK. La sincronizzazione avviene aggiungendo un offset al CLKN dello slave per farlo coincidere con il CLK della piconet.

-**CLKE**: Anche questo clock è derivato tramite un offset dal CLKN ed è usato dal master nel caso specifico della creazione di una connessione verso uno slave, e prima che tale slave si sia sincronizzato con il master (ovvero quando si tratta di un nuovo slave).

I primi 2 bit del contatore vengono usati direttamente per delimitare gli slot e i cosiddetti mezzi slot, per la trasmissione e ricezione dei pacchetti; essi servono anche a stabilire nel tempo gli slot Tx (trasmissione) o Rx (ricezione) a seconda che il dispositivo in questione stia funzionando da master o da slave. Una trasmissione da parte del master comincerà sempre quando CLK[1:0] = 00 (slot di indice pari), mentre una trasmissione da parte di uno slave comincerà sempre quando CLK[1:0] = 10 (slot di indice dispari).

### 3.3.2 Topologia delle reti

Due o più dispositivi collegati tra loro formano una piconet. I dispositivi all'interno di una piconet possono essere di due tipi: master o slave. Il master è il dispositivo che all'interno di una piconet si occupa di tutto ciò che concerne la sincronizzazione del clock degli altri dispositivi (slave) e la sequenza dei salti di frequenza. Gli slave sono unità della piconet sincronizzate al clock del master e al canale di frequenza. Le specifiche Bluetooth prevedono 3 tipi di topologie: punto-punto, punto-multipunto e scatternet. Diverse piconet possono essere collegate tra loro in una topologia chiamata scatternet. Gli slave possono appartenere a più piconet contemporaneamente, mentre il master di una piconet può al massimo essere lo slave di un'altra. Il limite di tutto ciò sta nel fatto che all'aumentare del numero di piconet aumentano anche il numero di collisioni dei pacchetti e di conseguenza degradano le prestazioni del collegamento. Ogni piconet lavora indipendentemente dalle altre sia a livello di clock che a livello di salti di frequenza. Questo perché ogni piconet

ha un proprio master. Un dispositivo bluetooth può partecipare sequenzialmente a diverse piconet come slave attraverso l'uso di tecniche TDM (Time Division Multiplexing), ma può essere master solo in una.

### 3.3.3 Modalità operative

-**Active mode**: L'unità partecipa attivamente alla piconet, sia in ricezione che in trasmissione, ed è sincronizzata al clock del master. Il master trasmette regolarmente per mantenere la sincronizzazione del sistema.

-**Hold mode**: Il master può mettere i dispositivi slave nello stato di Hold per un tempo determinato.Durante questo periodo nessun pacchetto può essere trasmesso dal master anche se il dispositivo mantiene la sincronizzazione con il master. Questa modalità operativa è utilizzata generalmente nel momento in cui non si devono inviare pacchetti ad un dispositivo per un periodo relativamente lungo (questo ci fa capire che tale modalità operativa è supportata solamente nel caso in cui tra due dispositivi bluetooth ci sia un collegamento di tipo ACL). Durante questo periodo, il dispositivo si può spegnere per risparmiare energia. L'Hold mode può essere utilizzata anche nel caso in cui un'unit vuole scoprire o essere scoperta da altri dispositivi bluetooth o vuole partecipare ad altre piconet.

-**Sniff mode**: Lo slave che passa in questo stato si trova in una modalità di risparmio energetico. Per entrare nello sniff mode, master e slave devono negoziare due parametri: uno sniff interval ed uno sniff offset. Con il primo si fissano gli slot di sniff, mentre con il secondo si determina listante del primo slot di sniff. Quando il collegamento entra in sniff mode, il master può inviare pacchetti solamente allinterno degli sniff slot. Quindi lo slave ascolta il canale ad intervalli ridotti. Il master pu costringere lo slave ad entrare in sniff mode, ma entrambi possono chiedere il passaggio. L'intervallo di sniff mode programmabile.

-**Park mode**: Questa modalit stata ideata per avere la possibilit di costituire piconet con pi di sette slave. Le unit in questo stato ascoltano regolarmente il traffico sulla rete per risincronizzarsi e ricevere messaggi di broadcast. Questi ultimi, infatti, sono gli unici messaggi che possono essere inviati ad uno slave in park mode. La richiesta di passaggio in park mode può avvenire indifferentemente da parte del master o dello slave.

### 3.3.4 Architetture

Per garantire l'interoperabilità fra dispositivi costruiti da produttori diversi, la specifica non definisce soltanto i requisiti del sistema radio, ma anche il protocollo di comunicazione. Esso è organizzato in una pila (stack), in cui ogni strato (layer), interagendo con i layer adiacenti, usa i servizi dei layer più in basso offrendone a sua volta a quelli superiori. In una comunicazione fra dispositivi Bluetooth ogni layer comunica col suo omologo presente su un altro dispositivo mediante il relativo protocollo, definito dalla specifica. Sono quindi presenti tante connessioni logiche quanti sono i layer utilizzati.

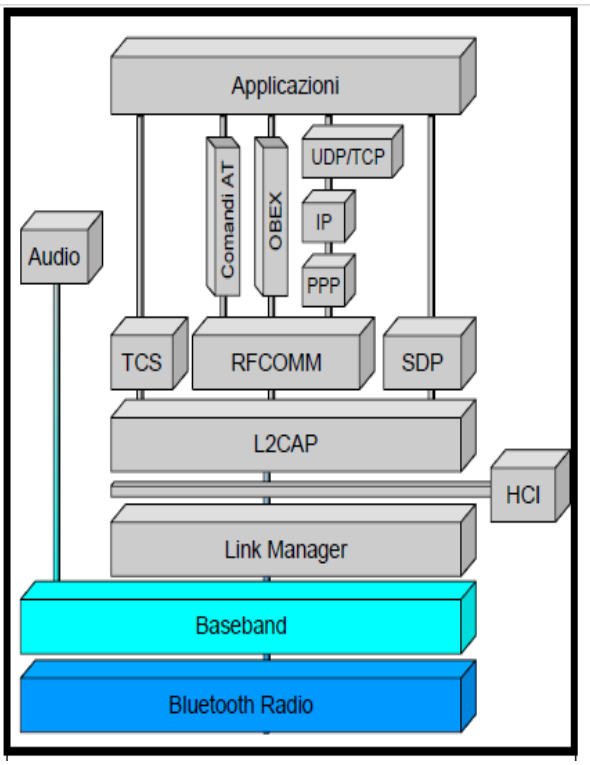


Figura 3.3: Architettura dispositivi BT

**-Baseband e Bluetooth Radio** : Sono gli strati più bassi e servono ad effettuare le procedure relative alla connessione fisica fra i dispositivi. In particolare stabiliscono la sequenza delle frequenze di salto e controllano il canale fisico, formano i pacchetti dati che verranno effettivamente trasmessi, implementano la correzione degli errori di trasmissione e l'eventuale cifratura dei dati.

-**Host Controller Interface (HCI)**: Non è un layer vero e proprio poichè non eroga servizi per i livelli superiori né comunica col suo omologo presente su un altro dispositivo. Serve piuttosto a fornire un'interfaccia uniforme di comandi per accedere al Link Manager e alle potenzialità del sistema radio. Offre, inoltre, accesso allo stato dell'hardware e ai registri di controllo.

- **Logical Link Control and Adaptation Protocol (L2CAP)**:Il layer L2CAP ha la funzione di fornire servizi per l'invio dei dati ai livelli superiori. A tale scopo, supporta:

- Multiplexing dei livelli più alti di protocollo;
- Segmentazione e riassemblaggio dei pacchetti di dimensioni elevate;
- Gestione delle connessioni logiche con i layer superiori;

Grazie a L2CAP, è consentito l'invio di pacchetti dati più grandi di quelli permessi dalla semplice trasmissione radio e l'invio di informazioni sulla qualità del servizio. Non è prevista la segmentazione di pacchetti audio sincroni ma soltanto di dati asincroni.

-**Service Discovery Protocol (SDP)**: Questo strato permette ad un dispositivo di individuare i servizi di cui potrebbe usufruire connettendosi ad altri dispositivi Bluetooth a portata di comunicazione e di notificare i servizi che possono essere forniti.

-**Telephony Control - Binary (TCS Bin)**:È un protocollo orientato ai bit, che fornisce funzionalità per il supporto di telefonia senza fili.

- **RFCOMM** : Permette di emulare porte seriali attraverso il layer L2CAP, con i relativi segnali. Per esempio, RFCOMM può emulare una porta con lo standard RS-232. [3] [7]

# Capitolo 4

# Protocollo Bluetooth NXT

Figura 4.1: Collegamento BT tra NXT

## 4.1 Introduzione

L' NXT non solo supporta la comunicazione Bluetooth tra programmi eseguiti su NXT differenti ma permette anche di comunicare con altri dispositivi Bluetooth quali Pc (Desktop o LapTop), cellulari ecc.. L'NXT può quindi ricevere svariati comandi tramite Bluetooth : attivare motori, effettuare letture dai sensori, avviare programmi, gestire file e molto altro anco-

ra. Di seguito vedremo le prestazioni e l'affidabilità di questa tipologia di comunicazione.

## 4.2 SPP Serial Port Profile

La comunicazione Bluetooth tramite NXT si basa su un protocollo chiamato SPP, Serial Port Profile. L'SPP emula il funzionamento di una porta seriale RS-232. Questo protocollo appartiene al livello Applicazione del modello ISO - OSI ed è implementato al di sopra di un protocollo di basso livello chiamato RFCOMM. RFCOMM è un protocollo di trasporto affidabile. Questo significa che i dati sono sempre consegnati a destinazione durante una comunicazione bluetooth, a meno che la connessione non venga interrotta. Gli sviluppatori del firmware NXT hanno costruito quindi un protocollo inaffidabile sopra un protocollo affidabile. Non affatto una scelta comune : in molti casi infatti, protocolli di alto livello affidabili vengono realizzati su protocolli di basso livello. Per esempio, TCP (protocollo affidabile), è costruito sopra IP (indaffidabile). La ragione di questa scelta è principalmente la memoria limitata dei dispositivi. Se il destinatario non estrae abbastanza velocemente i messaggi ricevuti dal buffer e il mittente vuole continuare la comunicazione, le scelte possibili sono due: scartare i messaggi in eccesso , oppure bloccare il mittente finché il destinatario non svuota il buffer. Gli sviluppatori NXT hanno deciso di non bloccare il dispositivo mittente.

## 4.3 Caratteristiche tecniche del protocollo

Più in dettaglio l'hardware Bluethooth è un chip a 16 bit che lavora ad una frequenza di clock pari a 26 MHz. Ha al suo interno installata una memoria RAM pari a 47 KByte e 8 Mbit di FLASH esterna. Il firmware installato è il Bluelab versione 3.1. La connessione con l'ARM7 è realizzata mediante l'interfaccia UART (a 8 bit, senza controllo parità e con bit di stop) e ha la possibilità di essere utilizzata in due modalità (approfondite nel paragrafo successivo), cioè per trasferire comandi o flussi di informazione. Entrambe le modalità permettono una velocità massima di 460 Kbit/s. La prima è utilizzata per istruzioni atte a creare la connessione, mentre la seconda viene utilizzata, a connessione avviata, per lo scambio dei dati. La tecnologia bluetooth implementata è la Class II con la quale si può comunicare con il brick ad una distanza di circa 10 metri. La velocità teorica massima per la trasmissione dati tra unità bluetooth è pari a 0.46 Mbit/s.

## 4.4 Master e Slave

La comunicazione bluetooth tra NXT classifica i dispositivi connessi in due tipologie : Master e Slave. Il Master è sempre colui che instaura la connessione Bluetooth. Se viene creata la connessione dal menu dell NXT, quell' NXT è il dispositivo Master. Se la connessione è stabilita con un PC o un altro NXT, il dispositivo che semplicemente accetta la richiesta di connessione è detto dispositivo Slave.

Il principio di funzionamento della relazione Master  Slave è molto semplice: solamente il Master, come già detto, inizia la comunicazione. La comunicazione può avvenire con richiesta di conferma di avvenuta ricezione (acknowledgement) o meno. Quando il dispositivo Master richiede dati dal dispositivo slave, la risposta contiene i dati se questi sono disponibili o un codice d'errore se non lo sono o se il programma sullo slave ha terminato la sua esecuzione.

Quando la connessione non è attiva, entrambe le parti coinvolte nella comunicazione, sanno che il passo successivo sarà sicuramente un messaggio dal Master. Successivamente, il dispositivo master specifica se necessita di risposta. Quando il dispositivo slave risponde (se la risposta era richiesta) la connessione viene disattivata e il processo si ripete. Questo meccanismo fa sí che in ogni istante, entrambi i dispositivi, sanno chi trasmetterà allo step successivo. Le modalità in cui possono trovarsi i dispositivi Master e Slave sono ben distinte, dunque, in Trasmissione e Ricezione. Il protocollo commuta tra le due a seconda delle esigenze. Non esistono modalità intermedie, le quali potrebbero causare incertezze su chi trasmette e chi riceve.

## 4.5 Mail Box

Quando un programma invia o riceve un messaggio, specifica una mailbox, identificata da un numero tra 0 e 9 (nel programma NXT-G da 1 a 10). Per lo slave, questo equivale alla costruzione di 10 canali virtuali che sottendono la comunicazione. L'utilizzo di queste mailbox da parte del dispositivo master non è banale. Un dispositivo NXT che funge da master pu essere connesso simultaneamente con 3 slave differenti. Quando il Master invia un messaggio, deve specificare a quale slave lo sta inviando e a quale mailbox è destinato. Quando il master riceve un messaggio, in esso è contenuta solo la mailbox relativa. Questa può dunque contenere un messaggio proveniente da uno qualsiasi dei tre dispositivi slave associato. Il dispositivo Master non può mai sapere da quale slave è stato inviato il messaggio, a meno che questo non sia esplicitato nel contenuto del messaggio.

## 4.6 Invio/ricezione dati Slave: System Call utilizzate

Il firmware dell'NXT utilizza principalmente 4 system call per la comunicazione program to program. Due sono utilizzate solamente dal dispositivo Master e sono **NXTCommBTCheckStatus** e **NXTCommBTWrite**. La prima controlla se il canale Bluetooth è occupato. Se lo è, il programma attende fino a che la NXTCommBTCheckStatus non riporta esito positivo. La seconda system call, NXTCommBTWrite, consegna il messaggio al canale per l'invio. Può essere invocata solo se il canale risulta libero. Il Firmware non gestisce code di messaggi che devono essere inviati dal dispositivo Master. Per ricevere un messaggio invece, il dispositivo Master invoca una terza system call, **NXTMessageRead**, e specifica da quale mail box effettuare la lettura. NXTMessageRead invia una richiesta al dispositivo slave affinché questo invii un messaggio alla specifica mailbox del dispositivo Master. Se il dispositivo Slave non risponde, la system call esaurirà il tempo d'attesa e si concluderà senza consegnare il dato al programma avviato sul dispositivo Master. Il programma avviato sul dispositivo slave utilizza a sua volta due system call per comunicare con il master. Quando il programma desidera ricevere un messaggio dal disposivio master, chiama la NXTMessageRead, la quale, se chiamata dallo slave, ha un comportamento differente: controlla semplicemente la mailbox specifica e ritorna il messaggio se lo trova. Non invia richieste al Master in quanto:

- Il dispositivo Slave non può iniziare una comunicazione;

- Il dispositivo Master invierebbe immediatamente comunque il messaggio perchè non contiene code in fase di trasmissione;

Per inviare un messaggio al dispositivo Master, lo slave invoca la **NXTMessageWrite**. Questa system call utilizzata solamente dai dispositivi slave. Aggiunge il messaggio ad una coda in una specifica mailbox. Il messaggio attenderà fino a quando master non lo trova (polling delle mail box).

## 4.7 Utilizzo di Mail Box multiple: Polling

I dati di tipo diverso (stringhe, numeri, booleani) possono essere inviati a mailbox differenti, permettendo in tal modo al dispositivo destinatario di determinare il tipo di dato ricevuto tramite il numero della mailbox. Anche dati di origine diversa (provenienti da sensori differenti per esempio) possono

essere inviati a mailbox differenti. Ma raggiungere prestazioni accettabili utilizzando questa tecnica risulta difficile. Nonostante per il dispositivo mittente non ci siano differenze tra inviare messaggi a mailbox diverse o alla medesima mailbox, c'è una grande differenza tra ricevere messaggi in una mailbox o in più mailbox. Quando il dispositivo master prova ad effettuare una lettura da mailbox multiple, deve effettuare il polling di tutte le mailbox. Ogni tentativo di polling impiega circa 30 ms. Questa operazione effettuata su mailbox vuote spreca un tempo non indifferente. A questo punto, è stato necessario effettuare una scelta: Effettuare un polling concorrente con più thread o esaminare sequenzialmente ogni mailbox.

## 4.8 Efficienza e prestazioni

Nonostante l'affidabilità del protocollo, il bluetooth NXT perde comunque dati durante la comunicazione tra master e slave. Ogni mailbox può contenere fino a 5 messaggi. Se un messaggio viene inviato ad una mailbox piena, il dato presente da maggior tempo nella coda, viene eliminato. Dunque, siccome i messaggi vecchi vengono semplicemente eliminati dalle code piene, non tutti i dati inviati vengono sempre consegnati. I messaggi che dal master transitano verso lo slave risiedono nelle code delle mail box di quest'ultimo il quale, se non effettua il polling abbastanza frequentemente, si troverà a far fronte ad un overflow, con conseguente perdita di dati. Allo stesso modo, se il programma avviato sul dispositivo slave continua ad inviare messaggi mentre il master non lo interroga abbastanza velocemente, i messaggi più vecchi ma non ancora consegnati verranno eliminati dalle mailbox. Nel peggiore dei casi, se il dispositivo slave invia più di 5 messaggi senza fermarsi, non c'é alcun modo per il dispositivo master, di ritrovarli tutti abbastanza velocemente. Lo slave manterrà nella sua Ram tutti i dati da inviare, ma il master avrà accesso (per il modo in cui è programmato il firmware) solamente al buffer delle mailbox.

### 4.8.1 Gestione dell'Overflow

Ci sono diverse tecniche che permettono di bypassare l'overflow delle code e ovviare così alla perdita di dati importanti durante la trasmissione. Tuttavia, alcune di queste soluzioni risultano non efficienti, altre adirittura non implementabili sul NXT, altre ancora non consentite per esempio nei programmi NXT-G. Un semplice modo per risolvere il problema è controllare che le code siano vuote prima di inviare un messaggio. Supponiamo che il dispositivo master e lo slave si alternino nell'inviare e ricevere dati. Ognuno

invia un messaggio, attende la conferma dell'arrivo a destinazione e solo successivamente ne invia un altro. Questo garantisce che ogni coda non contenga mai più di un messaggio, in modo tale da non generare mai overflow. Questa alternanza di invio-ricezione riduce notevolmente le prestazioni ottenibili dal protocollo bluetooth NXT. Il problema fondamentale risulta lo switch tra le due modalità, trasmissione e ricezione, che aumenta i tempi di comunicazione. Vediamo cosa potrebbe essere fatto per aumentare l'affidabilità dello scambio di dati senza grosse perdite di prestazioni. Una strada potrebbe essere quella di avere delle code di dimensioni maggiori. Tuttavia nell' NXT questo non è possibile perché il firmware è implementato e progettato per gestire code contenenti fino a 5-messaggi (perché la Ram è limitata). Ma se a comunicare con l'NXT è ad esempio un PC, su quest'ultimo possiamo utilizzare code molto ma molto più grandi. La tecnica è utile solamente se l'NXT funge da Master in quanto l'overflow si verifica nelle code delle mailbox contenute nello slave. Si rivela quindi inutile come soluzione se l'nxt svolge il ruolo di slave. Un'altra tecnica attuabile, quando l'NXT è il dispositivo Master, è quella di aspettare che il canale sia disponibile, prima di inviare un messaggio. Cioè che vengano consumati prima i dati già inviati, prima di una nuova trasmissione.

### 4.8.2 Sperimentazioni e Speed Test

Le misure effettuate per la determinazione delle prestazioni ed impostazioni ideali sono state realizzate con NXT che utilizzano il firmware v 1.6, programmato tramite un netbook packardbell con processore Intel Atom e sistema operativo win 7. Tutte le misure sono state effettuate in ambiente privo di altre comunicazioni bluetooth attive. L'esperimento consiste nel controllare il numero di dati che giungono a destinazione sotto opportune ipotesi. La prova è stata realizzata utilizzando 2 NXT : uno svolge sempre e solo il ruolo di mittente, l'altro di destinatario. Per l'implementazione del codice, sono state utilizzate le funzioni fornite nella libreria di Daniele Benedettelli, reperibile al link presente in bibliografia. Ecco le più importanti, utilizzate anche per la realizzazione del software di Molly e Beezer:

```
#ifndef CHANNEL
#define CHANNEL   1    //Il canale slave pu essere 1,2,3
#endif

#ifndef MAILBOX
#define MAILBOX   0    //0 a 9
#endif
```

```
#define MASTER    0    //Il canale master   sempre 0
#define v 14

byte __local_buffer[80];
byte __local_array[59];

int X[v];

void btwaitfor(int conn)
{
 byte e=NO_ERR+1;
 while(e!=NO_ERR)
 {
  e=BluetoothStatus(conn);
  if(e==NO_ERR) break;
  if(e==STAT_COMM_PENDING) continue;
  TextOut(0, LCD_LINE2, "Bluetooth error:", true);
  NumOut(30, LCD_LINE4, e);
  switch(e)
  {
   case ERR_COMM_CHAN_NOT_READY:
   TextOut(0, LCD_LINE6, "NXT bluetooth");
   TextOut(0, LCD_LINE7, "not connected!");
   break;

   case ERR_COMM_BUS_ERR:
   TextOut(0, LCD_LINE6, "bus error:");
   TextOut(0, LCD_LINE7, "please reboot");
   break;
  }
  Wait(7000);
  Stop(true);
 }

}

void sendtomaster(string msg)
{

 byte mbx=MAILBOX+10;
 btwaitfor(MASTER);
 SendMessage(mbx, msg);
 btwaitfor(MASTER);
```

```
53 }

55 void sendtoslave(string msg)
   {
57
    byte len;
59  int i;
    StrToByteArray(msg,__local_array);
61  len = ArrayLen(__local_array);

63  __local_buffer[0] = 0x80;    // Non richiede L'acknowledgment
    __local_buffer[1] = 0x09;    // MessageWrite Direct Command
65  __local_buffer[2] = MAILBOX; // Numero della MailBox
    __local_buffer[3] = len+1;   // Dimensione messaggio
67
    len=len+4;
69  i=4;

71  for(;;)
    {
73   __local_buffer[i] = __local_array[i-4];
     i++;
75   if(i>=len) break;
    }
77  btwaitfor(CHANNEL);
    BluetoothWrite(CHANNEL, __local_buffer);
79  btwaitfor(CHANNEL);

81 }


83
   string receivefrommaster()
85 {

87  string msg;
    btwaitfor(MASTER);
89  ReceiveMessage(MAILBOX, true, msg);
    btwaitfor(MASTER);
91  return msg;

93 }
```

```
95
   string receivefromslave()
97 {

99  string msg;
    btwaitfor(CHANNEL);
101 ReceiveMessage(MAILBOX, true, msg);
    btwaitfor(CHANNEL);
103 return msg;

105 }


107
   void btchannelcheck(int conn)
109 {

111 int e = BluetoothStatus(conn);
    string m;
113
    if(e==NO_ERR) return;
115 TextOut(0, LCD_LINE3, "Bluetooth error", true);
    NumOut(0,  LCD_LINE4, e);
117 TextOut(0, LCD_LINE8, "on channel -.");
    NumOut(66, LCD_LINE8, conn);
119
    if(conn==CHANNEL)
121 {
     TextOut(0, LCD_LINE1, "Master NXT");
123  TextOut(0, LCD_LINE6, "please connect");
     TextOut(0, LCD_LINE7, "the slave NXT");
125 }
    else
127 {
     TextOut(0, LCD_LINE1, "Slave NXT");
129  TextOut(0, LCD_LINE6, "please wait for");
     TextOut(0, LCD_LINE7, "the master NXT");
131 }
    Wait(11000);
133 Stop(true);

135 }

137 void mastercheck()  { btchannelcheck(CHANNEL); }
```

```
void slavecheck()   { btchannelcheck(MASTER);  }
```

Queste funzioni vengono richiamate all'interno del task main() per realizzare i test opportuni. Ecco il codice del dispositivo che svolge solo il ruolo di mittente :

```
task main()
{

 string t;
 //int X[];   Variabile globale

 for (i=0;i < v; i++)
 {
  X[i]=i;
 }

 for( s=0; s < v; s++)
 {
  ClearScreen();
  TextOut(0, LCD_LINE3,NumToStr(X[s]));
  Wait(1000);
  t=NumToStr(X[s]);
  sendtoslave(t);
 }

}
```

Come ben deducibile dal codice, il mittente non fa altro che creare un vettore di dimensione variabile ed inviarlo tramite bluetooth al dispositivo Slave associato. Codice ricevente:

```
void test_ricezione()
{

 string r, m, k;
 int i=0,j=0,g=0,xvx=0;

 //init vettore per controllo
 for(g=0;g<v;g++)
 {
  X[v]=g;
 }

```

```
    //NXT come dispositivo Slave
14  slavecheck();
    Wait(500);
16
    for(;i<=v;)
18  {
     //Ricevi dati da dispositivo Master
20   r = receivefrommaster();
     Wait(1000);
22
     //Lunghezza stringa ricevuta
24   j = StrLen(r);

26   //Tutto il resto viene fatto solo se   stato ricevuto qualcosa
     if(j!=0)
28   {
      TextOut(0,LCD_LINE3,r);
30    if (i > 3)
      {
32     k=NumToStr(X[i]);
       if(k != r)
34     {
        xvx++;
36      TextOut(0,LCD_LINE2,k);
        Wait(1000);
38     }
      }
40   }

42   //xvx variabile di controllo, se = 0 allora trasmissione ok
     if(xvx!=0)
44   {
      TextOut(0, LCD_LINE2, "ERRORE");
46   }
     else
48   {
      TextOut(0, LCD_LINE2, "OK");
50   }
     Wait(2000);
52
}
54
task main()
```

```
56 {

58  test_ricezione();

60 }
```

Come spiegato sopra, se il programma invia più di 5 messaggi sequenzialmente, tutti meno l'ultimo sono in genere scartati. Questa perdita di dati può essere aggirata attendendo 50 ms dopo l'invio di ogni messaggio. Inoltre, se la comunicazione termina con un messaggio inviato da parte dello slave, diventa indispensabile attendere dopo averlo spedito. L'esperimento in cui l'NXT invia 1000 messaggi sequenzialmente, come previsto, ha causato la perdita di quasi tutti i messaggi. Nonostante le diverse prove effettuate, non si è riuscito ad ottenere un risultato migliore di 40 messaggi consegnati, sui 1000 inviati. Tutti gli altri vengono eliminati dalle code delle mailbox quando si verifica overflow. Inserire il delay di 50 ms dopo l'invio di ogni dato garantisce una efficenza di quasi il 100 % a scapito delle prestazioni.

## 4.9 Conclusioni

In base ai test effettuati e successivamente ad approfondimenti specifici e mirati, le conclusioni finali sul bluetooth NXT sono le seguenti:

- Il protocollo bluetooth NXT risulta non affidabile, per scelta di progettazione. È molto facile perdere messaggi durante la trasmissione, quindi, se il progetto da realizzare richiede maggiore affidabilità, bisogna implementrare un opportuno protocollo.

- Se sono richieste prestazioni superiori vi sono due vie percorribili. Se è richiesta maggior velocità di trasmissione, configurare l'NXT come master e spedire ad un dispositivo che non presenta problemi di overflow, per esempio un pc. Se i messaggi sono di dimensioni ridotte, otteniamo un buon livello di affidabilità anche nell'utilizzare l'ambiente di sviluppo NXT- G (nonostante i diversi timeout presenti nel blocco di trasmissione nxt-g). Se i messaggi sono di dimensioni maggiori, conviene invece utilizzare ambienti di sviluppo che non prevedono timeout quando il canale è occupato per una trasmissione.

- Se si necessita di acquisire dati nell'NXT velocemente, configurarlo come slave e spedire i dati senza richiesta di acknowledgement. L'aspettativa è logicamente quella di perdere un certo numero di dati, ma i messaggi consegnati giungeranno a destinazione in meno di 5 ms per messaggio. Non c'è modo di ottenere la stessa velocità di trasmissione senza perdere affidabilità.

- Risulta molto difficile realizzare un programma affidabile che utilizzi mailbox multiple.

Una soluzione valida per un incremento notevole di affidabilità potrebbe essere quello di modificare il firmware di modo che i dati vengano inviati specificandole la priorità, di modo che dati di importanza fondamentale non vengano mai persi. Questo implica tuttavia che il mittente possa essere bloccato, per periodi di tempo indefiniti. [1] [3] [2] [7]

# Capitolo 5

# NXTCam V2

Figura 5.1: NXTCam V2

## 5.1 Introduzione

Per la realizzazione del progetto, è stato utilizzata un NxtCam V2. NXT-CAM è un motore di elaborazione immagini in real-time con processore on-board, interfacciabile con l'NXT attraverso le comuni porte I2C. Non compresa nel kit standard di Lego NXT, permette l'acquisizione di informazioni ad alto livello e processabili successivamente, riguardanti le immagini inquadrate. Le informazioni processate, contengono le coordinate dell'oggetto che interessa l'NXT-Cam. L'NxtCam infatti, non invia l'immagine all'NXT, tuttavia, è possibile vedere cosa essa rileva, collegandola via USB ad un PC ed utilizzando il software di configurazione.

## 5.2 Caratteristiche Tecniche

NXTCam presenta le seguenti caratteristiche tecniche :

- Rileva fino a 8 colori differenti;
- Configurabile tramite Usb con Windows XP, Vista, 7;
- 2 modalità di acquisizione: Object tracking e Line tracking;
- Immagini acquisite con risolzione di 88 x 144 pixel a 30 frame/secondo;
- Massimo consumo 42 mA a 4.7 V;
- Compatibile con I2C;

## 5.3 Antenato AVRCam

Figura 5.2: AVRCam

Il firmware dell'NXTCam, si basa interamente su un progetto open source chiamato AVRCam. Il vantaggio sostanziale dell'AVRCam è la possibilità di operare modifiche radicali grazie al fatto che è completamente OpenSource. Le caratteristiche tecniche sono pressoché identiche all'NXTCam, ma non essendo proprietaria, è possibile l'alterazione di queste con una scelta differente dei componenti ed una riprogrammazione di alcune funzionalità. Al seguente link è presente una guida passo passo per la realizzazione del AvrCam

Figura 5.3: NXTCam V3

## 5.4 Confronto con NXT V3

Con la versione V3 del sensore, Mindesensors ha reso più semplice l'aggiornamento del microcontrollore integrato e, allo stesso tempo, è stato rilasciato il codice sorgente e gli strumenti necessari per modificarlo. Ora, l'unica cosa a cui non si può mettere mano, è la porzione di codice che consente il collegamento della Cam alla porta per sensori dell NXTCam.

## 5.5 Utilizzo

### 5.5.1 Collegamento

La NXTCam può essere connessa ad una qualunque delle porte per sensori dell'NXT utilizzando il cavo standard di connessione. Nel programma poi, basterà far attenzione a impostare il giusto numero di porta.

ATTENZIONE!!: Non collegare assolutamente l'NXTCam alle porte per motori, perché il voltaggio applicato per quest'ultimi danneggerebbe l'NXTCam.

### 5.5.2 Driver

Affinché l'NXTCam funzioni correttamente, è necessario installare l'USB driver corretto a seconda del sistema operativo utilizzato. Attualmente sono supportati i seguenti sistemi operativi

- Windows XP (i386 and AMD processors);

- Windows Vista (i386 and AMD processors);
- Mac OS X v10.4.10 (PowerPC G4);

I driver appropriati possono essere scaricati, assieme alle istruzioni di istallazione dal seguente link:

`http://www.mindsensors.com/index.php?module=pagemaster&PAGE_user_op=view_page&PAGE_id=78`

I test e l'implementazione finale dell'intero lavoro, sono stati effettuati configurando l'Nxt Cam sia con Windows Xp, Windows 7 (stessi driver di win Vista) e Mac Os.

### 5.5.3 NXTCam View

Per visualizzare il campo visivo dell'NxtCam, acquisire l'immagine per analizzare e configurare la Colormap per processi onboard, è necessario installare e utilzzare il software di configurazione appropriato sul proprio pc. Per MS-Windows XP/ Vista e dai test effettuati anche MS-Windows 7,scaricare il software da :

`http://nxtcamview.sourceforge.net/`

Per Machintosh, scaricare il software da :

`http://www.mindsensors.com`

Pur essendo la configurazione su sistema Windows più intuitiva, quella su Mac Os è risultata essere più precisa nella scelta della colormap.

### 5.5.4 ColorMap

Gli oggetti interessati vengono riconosciuti dalla NXTCam confrontando il valore del colore rilevato con quello precedentemente memorizzato. Per poter far ciò è necessario quindi che i valori dei colori degli oggetti interessati, siano memorizzati all'interno della NXTCam. I valori di questi colori compongono la cosidetta Colormap. L'NXTCam può memorizzare fino a 8 colori differenti e fornisce le informazioni processate eseguendo il confronto con queste Colormap.

### 5.5.5 Condizione di luce e messa a fuoco

Vediamo una breve illustrazione dettagliata su come funziona il sistema di acquisizione ottico di una fotocamera per capire meglio il problema della luminosità (sicuramente uno dei problemi più insidiosi dell'esperienza). Prima di tutto è importante sapere che le immagini sono convertite in segnali elettrici mediante sensori ottici. Il campionamento spaziale e la quantizzazione del segnale elettrico danno origine ad un'immagine digitale. Il ruolo principale nell'acquisizione dell'immagione è svolto dal sensore ottico. Si tratta solitamente di un sensore CCD (dispositivo a trasferimento di carica). Ogni fotoelemento è costituito da un dispositivo al Silicio. I fotoni giunti al fotoelemento, attraversano una struttura di gate di Si policristallino trasparente e vengono assorbiti dallo strato interno (sempre di Si), dando origine a coppie elettrone-lacuna. La carica elettrica generata, in ogni fotoelemento, è proporzionale all'intensità luminosa focalizzata in quel punto. Il numero di fotoelementi ovvero il numero di valori d' intensità luminosa acquisiti (nxm) definisce la risoluzione dellimmagine ottenuta, es. 640x480. Tali valori (quantizzati) vengono poi inviati e memorizzati per essere successivamente elaborati.

Per quanto riguarda le diverse tonalità di colore invece, mentre l'occhio umano, insieme al cervello, à in grado di interpretare come uniformemente bianche le varie sfumature di colore emesse da differenti tipi di luce (lampada alogena, tubo al neon, lampadina a bulbo, luce naturale), la fotocamera percepisce le diversità nel colore della luce proveniente da sorgenti diverse e non ha la capacità di convertirle in luce bianca, se non con lintervento di opportuni comandi di bilanciamento del bianco o nel nostro caso, di scelta di un colore opportuno in base alle necessità di rilevazione.

Due sono le situazioni fondamentali di luce: interni o indoor, esterni o outdoor. In ogni frangente, il colore della luce ha uno stretto rapporto con la sua temperatura, tanto che si parla proprio di temperatura colore, misurata in gradi Kelvin (la luce del giorno, ad esempio, misura mediamente 5.600 gradi Kelvin). Sebbene sia interpretata dal cervello come bianca, essa ha precisi contenuti cromatici, che possono variare anche velocemente, da una predominanza di rosso ad una di blu, con il mutare della posizione del sole. Così, dall'alba al tramonto, la luce cambia continuamente colore e temperatura.

Ecco dunque il problema principale dell utilizzo della NXT Cam: il cambiamento continuo delle condizioni luminose.

L'NXTCam è stata progettata per operare sotto luci fluorescenti. Si può dunque ridurre disturbi luminosi proveniente ad esempio da sorgenti luminose vicine all'infrarosso effettuando le misure di interesse sotto illuminazioni

fluorescenti. Tuttavia, per operazioni avanzate, è possibile cambiare le impostazioni del colore, il contrasto e la luminosità dell NXTCam alterando i valori registri del I2C. Le impostazioni di fabbrica fanno sí che le lenti della NXTCam abbiano una messa a fuoco ottimale a circa un metro di distanza. È possibile tuttavia aggiustare manualmente la messa a fuoco ruotando leggermente la lente. Durante questa operazione è consigliabile utilizzare il software Viewer per testare l'impostazione desiderata, per vedere se il risultato è soddisfacente.

### 5.5.6 Consigli utili per l'utilizzo dell'NXTCam

L'deale è operare dunque in un ambiente privo di variazioni luminose frequenti, quindi possibilmente indoor, magari illuminando il percorso di interesse per il robot con delle lampade al neon. Tuttavia, non essendo questo sempre possibile, è importante poter modificare subito la colormap per un elaborazione utile del segnale acquisito tramite la fotocamera. È dunque indispensabile effettuare ricalibrazioni costanti della colormap, effettuando scansioni periodiche dell immagine percepita dall' NXT Cam, per scegliere (nel caso specifico del nostro esperimento) un colore con maggior contrasto rispetto all'ambiente circostante ([1] [2] [4] [5]). È possibile inoltre eseguire prima di ogni altro programma, queste righe di codice di test, per vedere se l'oggetto del colore interessato è rilevato correttamente :

```
const byte camPort  =  IN_1;

#define CAMADDR     0x02
#include "nxtcamlib.nxc"

task main()
{
 int nblobs;
 int stop_now = 0;
 int bc[10];
 int bl[10];
 int bt[10];
 int br[10];
 int bb[10];
 int init, n, i;
 int center_x, center_y;
 string msg, topdown, leftright;

 init = NXTCam_Init(camPort, CAMADDR);

```

```
  while (stop_now == 0)
22 {
   TextOut(40, LCD_LINE1, "", true);
24  n ++;
   NXTCam_GetBlobs(camPort, nblobs, bc, bl, bt, br, bb);
26        if ( nblobs > 0 )
   {
28   center_x = (bl[0] + br[0])/2;
   center_y = (bt[0] + bb[0])/2;
30   msg = "CX ";
   msg += NumToStr(i);
32   msg += ": ";
   msg += NumToStr(center_x);
34   TextOut(40, LCD_LINE1, msg, false);
   msg = "CY ";
36   msg += NumToStr(i);
   msg += ": ";
38   msg += NumToStr(center_y);
   TextOut(40, LCD_LINE2, msg, false);
40   msg = "                ";
   TextOut(0, LCD_LINE7, msg, false);
42
   /* cam center is at 44 x 72 */
44         topdown = "stay put";
   leftright = "stay put";
46   if (( center_y < 54 || center_y > 34 )||
   ( center_x < 82 || center_x > 62 ))
48   {
    topdown = "AAAAAAAAAAAAA";
50   }
   TextOut(0, LCD_LINE5, topdown, false);
52   Wait(400);
   } else
54   {
    TextOut(0, LCD_LINE5, "              ", false);
56    TextOut(0, LCD_LINE6, "              ", false);
    msg = "Nessun oggetto trovato";
58    TextOut(0, LCD_LINE7, msg, false);
    Wait(400);
60   }
        }
62 }
```

### 5.5.7 Codice Bezeer : Gestione NXTCam

Ecco il codice realizzato per effettuare la lettura del colore del fiore da parte di Beezer :

```
/*
 * Codice Beezer NXTCam
 * Autori : Franco Marangoni & Zahir Fetovski
 *
*/

//Libreria NXTCam
#include "nxtcamlib.nxc"

//Subroutine della telecamera
sub telecamera()
{
 int find=0;
 int nblobs;
 int stop_now = 0;
 int bc[10];
 int bl[10];
 int bt[10];
 int br[10];
 int bb[10];
 int init, n, i, p, ni;
 int center_x, center_y;
 string msg, topdown, leftright;
 init = NXTCam_Init(camPort, CAMADDR);
 /*
 I colori sono predefiniti nella NXTCam.
 1 e 5 sono i colori del fiore
 0 2 e 4 i colori del veleno o dell'ambiente
 */
 ni=0;
 while ((stop_now == 0)&&(ni<8))
 {
  ni++;
  n ++;
  NXTCam_GetBlobs(camPort, nblobs, bc, bl, bt, br, bb);
  if ( nblobs > 0 )
  {
   msg="";
   ClearScreen();
```

```
40      Wait(500);
        for(i=0; i<8;i++)
42      {
         if((bc[i]==1)||(bc[i]==5))
44       {
          find=1;
46        stop_now=1;
          i=8;
48       }
         else
50       {
          if((bc[i]==0)||(bc[i]==2)||(bc[i]==4))
52        {
           find=2;
54         stop_now=1;
           msg+=NumToStr(bc[i]);
56        }
                TextOut(0, LCD_LINE1, msg, true);
58        Wait(1000);
          else
60        {
           ni++;
62        }
         }
64       if(find==1)
         {
66        avanza(300);
          ClearScreen();
68
          //Visualizzo i dati inizializzati sul display
70        TextOut(0,LCD_LINE1,"FIORE");
          Wait(1000);
72        arriv=1;
          smoke();
74        Wait(3000);
          bluetooth();
76        Wait(2000);
          StopAllTasks();
78       }
               else if(find==2)
80       {
          ClearScreen();
82              TextOut(0, LCD_LINE1, "VELENO", true);
```

```
            PlayToneEx(440,1000,VOL,FALSE);
84          PlayToneEx(444,1000,VOL,FALSE);
            Wait(1000);
86      }
          else if(find==0)
88      {
        ClearScreen();
90          TextOut(0, LCD_LINE1, "VUOTO", true);
            PlayToneEx(440,1000,VOL,FALSE);
92        PlayToneEx(444,1000,VOL,FALSE);
                Wait(1000);
94          }

96  }
```

# Capitolo 6

# Allegati

Al fine di poter riprendere in mano tutto il lavoro svolto assieme al collega Franco Marangoni, invito alla lettura della sua tesi per avere un quadro completo sugli algoritmi utilizzati. Allego il codice completo dei tre robot realizzati, che potrebbero tornare utili per un eventuale progetto futuro con il kit Mindstorm.

## 6.1 Codice completo Robot Bezeer

```
/*
*
* Beezer - ape ricercatrice
* NXC code
* versione: beta 4.3
* autore: Franco Marangoni, Zahir Fetovski
* Ultime modifiche 19/05/11
*
*/

//Distanza ruote 11.0 cm
#define DISTANZA_RUOTE 110

//Raggio ruote 2.8 cm
#define RAGGIO_RUOTE 28

//Potenza Motori
#define POWER 60

//Lunghezza di ogni cella della griglia 40.0 cm
```

```
#define LUNGHEZZA_CELLA 400

//Distanza minima in cm a cui deve trovarsi l'ostacolo
//per far cambiare percorso al robot
#define DIST_MIN_OSTACOLO 55

//Valore limite di luminosita' che identifica il colore
//bianco dal colore verde
#define SOGLIA_LUCE 50

//Definizione di un intervallo di luminosita' che
//identifica il colore bianco dal colore verde
#define SOGLIA_LUCE_MIN 53
#define SOGLIA_LUCE_MAX 56
#define ASCII_ZERO 48
#define VOL 7
#define T_MIN 650
#define T_MAX 800

//Quanti controlli deve effettuare durante le rotazioni
#define TAPPE 5

const byte camPort  =  IN_1;

#define CAMADDR     0x02
#include "nxtcamlib.nxc"

/*
mappa[] contiene la struttura della mappa:
mappa[ y*b + x] contiene la casella c[x][y] della griglia:
mappa[ y*b + x]==1 --> cella con ostacolo
mappa[ y*b + x]==0 --> cella libera
*/

//Variabile che rappresenta la mappa
byte mappa[];

//Vettore delle distanze di ogni cella dal goal
int dist[];

//Variabile ausiliaria per segnalare errori di inizializzazione
int err;
```

```
64 //x,y goal
   int xg, yg;
66
   //Posizionamento del robot nella griglia
68 int xr,yr;

70 //Orientazione del robot
   int verso;
72
   //Area della griglia
74 int area;

76 //Base e altezza della griglia
   int b, h;
78
   int nord=-1, sud=-1, est=-1, ovest=-1, no=-1;
80 int ne=-1, so=-1, se=-1, arriv=0, global=0;


82
   // BLUETOOTH //
84


86 //Codice allegato nella sezione Bluetooth NXT

88 void bluetooth()
   {
90
    string a, b, c, t1, t2, t3, t4, m;
92  int io = 1, n = 0, s=0;
    int Xlen = ArrayLen(X);
94  for(;io<Xlen+1;)
    {
96   Wait(1000);
     t1="";
98   for( s=io-1; s<(5+io-1); s++)
     {
100   t2=NumToStr(X[s]);
      t1 = StrCat(t1, t2);
102  }
     //Invia messaggio all' NXT slave:
104  sendtoslave(t1);
     Wait(NAP);
106  ResetSleepTimer();
```

```
    io=io+5;
108  }

110 }

112 // FINE BLUETOOTH

114 //Avanza di dist in mm
    sub avanza(long dist)
116 {

118  long k;
     Off(OUT_BC);
120  k = dist*36000/(628*RAGGIO_RUOTE);
     RotateMotorEx(OUT_BC, POWER, k, 0, true, true);
122
    }
124
    // Controllo di non avere ostacoli davanti
126 int controllo(int ostacolo)
    {
128  if (SensorUS(IN_4) < DIST_MIN_OSTACOLO)
     {
130   PlayTone(440,100);
      PlayTone(220,100);
132   ClearScreen();
      TextOut(0,LCD_LINE1,"Rilevato Ostacolo");
134   TextOut(0,LCD_LINE2,"Ricalcolo percorso...");
      ostacolo=1;
136  } else
       {
138     ostacolo=0;
       }
140    return ostacolo;
    }
142
    //Gira a destra di x gradi (fa perno sulla ruota destra)
144 sub gira_dx(long x)
    {
146
     long k;
148  int i=TAPPE;
     //Variabile ausiliaria che segna errore se supero la linea
```

```
150  err=0;
     Off(OUT_BC);
152  k =  (DISTANZA_RUOTE*11*x/(RAGGIO_RUOTE*10));
     k/=TAPPE;
154  while ((Sensor(IN_3)<SOGLIA_LUCE ) && i-->0)
     RotateMotor(OUT_B, POWER, k);
156  //Se ho incontrato la linea lo segnalo
     if (i>0) err=1;
158  while (i-->0) RotateMotor(OUT_B, POWER, k);
     Off(OUT_BC);
160  Wait(500);
     if(err==1) avanza(-(200));
162
   }
164

166 //Gira a sinistra di x gradi (fa perno sulla ruota sinistra)
   sub gira_sx(long x)
168 {

170  long k;
     int i=TAPPE;
172  err=0;
     Off(OUT_BC);
174  k =  (DISTANZA_RUOTE*11*x/(RAGGIO_RUOTE*10));
     k/=TAPPE;
176  while ((Sensor(IN_3)<SOGLIA_LUCE) && i-->0)
     RotateMotor(OUT_C, POWER, k);
178  if (i>0) err=1;
     while (i-->0) RotateMotor(OUT_C, POWER, k);
180  Off(OUT_BC);
     Wait(500);
182  if(err==1) avanza(-(200));

184 }

186 //Gira di 180 gradi
   sub gira_180()
188 {

190  gira_sx(90);
     gira_dx(-90);
192  RotateMotor(OUT_BC, POWER, DISTANZA_RUOTE);
```

```
194 }

196 //Legge un numero da file
    int read_number(byte handle)
198 {

200  int i, ris=0;
     string s;
202  ReadLnString(handle, s);
     return StrToNum(s);
204
    }
206
    //Suona smoke on the water
208 sub smoke()
    {
210
     PlayToneEx(329,400,VOL,FALSE); Wait(600);
212  PlayToneEx(392,400,VOL,FALSE); Wait(600);
     PlayToneEx(440,700,VOL,FALSE); Wait(700);
214  PlayToneEx(329,400,VOL,FALSE); Wait(600);
     PlayToneEx(392,400,VOL,FALSE); Wait(600);
216  PlayToneEx(466,400,VOL,FALSE); Wait(400);
     PlayToneEx(440,700,VOL,FALSE); Wait(700);
218  PlayToneEx(329,400,VOL,FALSE); Wait(600);
     PlayToneEx(392,400,VOL,FALSE); Wait(600);
220  PlayToneEx(440,600,VOL,FALSE); Wait(600);
     PlayToneEx(392,300,VOL,FALSE); Wait(600);
222  PlayToneEx(329,700,VOL,FALSE); Wait(1000);

224 }


226
    //Posizione attuale nella mappa
228 sub ora()
    {
230
     int now = xg+yg*b;
232  ClearScreen();
     TextOut(0, LCD_LINE1, NumToStr(now), true);
234  int p=0;
     if(now>4)
```

```
236  {
      nord=now-5;
238   if(now%5!=0)
      no=now-5-1;
240   if((now+1)%5!=0)
      ne=now-5+1;
242  }
     if(now<45)
244  {
      sud=now+5;
246   if(now%5!=0)
      so=now+5-1;
248   if((now+1)%5!=0)
      se=now+5+1;
250  }
     if(now%5!=0)
252  ovest=now-1;
     if((now+1)%5!=0)
254  est=now+1;

256 }

258 //Subroutine NXTCam : Vedi codice allegato sezione NXTCam

260 //Riallineamento del robot
    sub riallinea()
262 {

264  int i1, i2, i3, is;
     int  ost_sx=0, ost_dx=0;
266  int ost_front=0, ost_front1=0,ost_front2=0;
     int aspetta, rand=0;
268  avanza(40);
     i1=0;
270  i2=0;
     is=0;
272
     //Avvio cronometro per il calcolo del tempo
274  i3 = CurrentTick();

276  //Inizio a girare
     OnFwd(OUT_C,POWER/2);
278  OnRev(OUT_B,POWER/2);
```

```
    while (Sensor(IN_3)<SOGLIA_LUCE) ;
280
    // Quando trovo la linea mi fermo
282 Off(OUT_BC);

284 // Calcolo del tempo
    i1=CurrentTick()-i3;
286 Wait(500);
    OnFwd(OUT_B,POWER/2);
288 OnRev(OUT_C,POWER/2);
    Wait(i1);
290 Off(OUT_BC);
    Wait(500);
292
    //Avvio cronometro per il calcolo del tempo
294 i3 = CurrentTick();

296 //Inizio a girare
    OnFwd(OUT_B,POWER/2);
298 OnRev(OUT_C,POWER/2);
    while (Sensor(IN_3)<SOGLIA_LUCE) ;
300
    //Quando trovo la linea mi fermo
302 Off(OUT_BC);

304 // Calcolo del tempo
    i2=CurrentTick()-i3-1;
306 Wait(200);
    i3=i1+i2;
308 i3=i3/2;

310 //Riallineo con tempo medio tra i due
    OnFwd(OUT_C,POWER/2);
312 OnRev(OUT_B,POWER/2);
    Wait(i3);
314 Off(OUT_BC);
    Wait(200);
316
}
318
//Mi sposto nel quadrato successivo
320 sub passetto()
{
```

```
322
    OnFwdReg(OUT_BC,POWER,OUT_REGMODE_SYNC);
324  int t1=0, tlight;
    while (true)
326  {
     tlight = Sensor(IN_3);
328   //Entro in linea bianca
     if ((tlight>SOGLIA_LUCE_MIN &&
330   tlight<SOGLIA_LUCE_MAX) && t1==0)
     {
332    t1=1;
      PlayTone(440,100);
334   }
     //Esco da linea bianca
336   else if ((tlight<SOGLIA_LUCE_MIN ||
     tlight>SOGLIA_LUCE_MAX) && t1==1)
338   {
      t1=0;
340    PlayTone(220,100);
      Off(OUT_BC);
342    break;
     }
344  }
    riallinea();
346  avanza(200);

348 }

350 //Calcolo tramite wave front algorithm dell distanze
   sub wave_front_algorithm()
352 {

354  string st2="", st3="", st1="", st4="",
    st5="", st6="", st7="";
356  int i,j,t;
    byte file;
358  string s1,s2;
    i = xg+yg*b;
360  //Se sono arrivato
    if (mappa[i] == 1)
362  {
     telecamera ();
364  }
```

```
    for (i=0; i<area; i++) dist[i] = area;
366 //Wave front algorithm
    st2 = NumToStr(yg);
368 st3 = NumToStr(xg);
    st1 = StrCat("Goal: ",st2,", ",st3);
370 TextOut(0,LCD_LINE5,st1);
    dist[yg*b+xg] = 0;
372 //Considero celle distanti i
    for (i=0; i<area; i++)
374 {
     //Considero tutte le caselle
376  for (j=0; j<area; j++)
     {
378   if (dist[j]==i)
      {
380    if (j-b>0 && mappa[j-b]==0 && dist[j-b]>i)
       dist[j-b]=i+1;
382    if (j+b<area && mappa[j+b]==0 && dist[j+b]>i)
       dist[j+b]=i+1;
384    if (j>0 && (j%b)!=0 && mappa[j-1]==0 && dist[j-1]>i)
       dist[j-1]=i+1;
386    if (j<area-1 && ((j+1)%b)!=0 && mappa[j+1]==0 &&
       dist[j+1]>i)
388    dist[j+1]=i+1;
      }
390   if (dist[yr*b+xr] < area) break;
     }
392  if (dist[yr*b+xr] < area) break;
     //Se arrivo dal robot -> fine
394 }
    st7=NumToStr(dist[xr+yr*b]);
396 st7 = StrCat("DISTANZA: ",st7);
    ClearScreen();
398 TextOut(0,LCD_LINE1,st7);
    //Goal impossibile da raggiungere
400 if (dist[yr*b+xr] == area)
    {
402  ClearScreen();
     TextOut(0,LCD_LINE1,"Impossibile raggiungere");
404  TextOut(0,LCD_LINE2,"il goal");
     PlayTone(330,200);
406  Wait(4000);
     err=1;
```

```
408     StopAllTasks();
        }
410     if((dist[xr+yr*b]<1))
        {
412      switch(verso)
         {
414       case 0:
          yr--;
416       break;

418       case 1:
          yr++;
420       break;

422       case 2:
          xr++;
424       break;

426       case 3:
          xr--;
428       break;
         }
430      telecamera();
         switch(verso)
432      {
          case 0:
434       yr++;
          break;
436
          case 1:
438       yr--;
          break;
440
          case 2:
442       xr--;
          break;
444
          case 3:
446       xr++;
          break;
448      }

450     }
```

```
 st4 = NumToStr(yr);
 st5 = NumToStr(xr);
 st6 = StrCat("Robot: ",st4,", ",st5);
 st7=NumToStr(dist[xr+yr*b]);
 st7 = StrCat("DISTANZA: ",st7);
 ClearScreen();
 TextOut(0,LCD_LINE1,st7);
 TextOut(0,LCD_LINE2,st6);
 TextOut(0,LCD_LINE3,st1);
 TextOut(0,LCD_LINE4,NumToStr(verso));
 Wait(1000);
 err=0;

}

//Carica la mappa da file
sub crea_path()
{

 byte handle_in;
 int fsize;
 int i,j;
 string s, st1,st2,st3;
 int t,t2;

 //Esce se c'  errore in apertura file, esci
 if(OpenFileRead("map.txt", fsize, handle_in) != NO_ERR)
 {
  ClearScreen();
  TextOut(0,LCD_LINE1,"Impossibile aprire");
  TextOut(0,LCD_LINE2,"map.txt");
  PlayTone(330,200);
  Wait(4000);
  return;
 }

 //Carico base e altezza
 b = read_number(handle_in);
 h = read_number(handle_in);
 area = b*h;
 ArrayInit(dist,0,area);
 ArrayInit(mappa,0,area);
 for (i=0; i<h; i++)
```

```
494  {
      if (ReadLnString(handle_in, s) == NO_ERR)
496   for (j=0; j<b; j++)
      {
498    t = StrIndex(s, j);
       if (t==ASCII_ZERO+1)
500    {
        //Colloco ostacolo in mappa
502     mappa[i*b+j]=1;
       }
504    if (t==ASCII_ZERO+2 || t==ASCII_ZERO+3 ||
       t==ASCII_ZERO+4 || t==ASCII_ZERO+5)
506    {
        xr = j; yr = i; verso=t-ASCII_ZERO-2;
508    }
       //x e y del goal
510    if (t==ASCII_ZERO+6)
       {
512     xg = j; yg = i;
       }
514   }
     }
516  ClearScreen();
     TextOut(0,LCD_LINE1,"base: ");
518  NumOut(50,LCD_LINE1,b);
     TextOut(0,LCD_LINE2,"altezza: ");
520  NumOut(50,LCD_LINE2,h);
     st1 = "Robot in ";
522  st2 = NumToStr(yr);
     st3 = NumToStr(xr);
524  st1 = StrCat(st1,st2,", ",st3);
     TextOut(0,LCD_LINE3,st1);
526  st1 = NumToStr(verso);
     st2 = StrCat("Orientazione: ",st1);
528  TextOut(0,LCD_LINE4,st2);
     st2 = NumToStr(yg);
530  st3 = NumToStr(xg);
     st1 = StrCat("Goal: ",st2,", ",st3);
532  TextOut(0,LCD_LINE5,st1);
     wave_front_algorithm();
534  CloseFile(handle_in);

536 }
```

```
538 /*
    Prima di muoversi da una cella all'altra il robot
540 si gira nel verso di marcia corretto, inoltre aggiorna
    le variabili di posizione allo stato che si trovera'
542 dopo il movimento. Eventualemente se vede ostacoli n
    nella direzione prevista,li memorizza, ricalcola
544 il percorso.
    */
546
    sub orienta_correttamente()
548 {

550  int ostacolo=0,t1;
     int gt=0, l=xg+yg*b;
552  int re;
     err=0;
554  do
     {
556   t1 = xr+yr*b;
     //Prosegue dritto
558  if ((verso==0 && yr>0 && mappa[t1-b]==0 &&
     dist[t1-b]==dist[t1]-1) ||(verso==1 && yr<h-1 &&
560  mappa[t1+b]==0 && dist[t1+b]==dist[t1]-1) ||
     (verso==2 && xr<b-1 && mappa[t1+1]==0 &&
562  dist[t1+1]==dist[t1]-1) ||(verso==3 &&
     xr>0 && mappa[t1-1]==0 && dist[t1-1]==dist[t1]-1))
564  {
      X[contatoreX]=1;
566   contatoreX++;
     }
568  //Gira a sinistra
     else if ((verso==0 && xr>0 && mappa[t1-1]==0 &&
570  dist[t1-1]==dist[t1]-1) ||(verso==1 && xr<b-1 &&
     mappa[t1+1]==0 && dist[t1+1]==dist[t1]-1) ||
572  (verso==2 && yr>0 && mappa[t1-b]==0 &&
     dist[t1-b]==dist[t1]-1) ||(verso==3 && yr<h-1 &&
574  mappa[t1+b]==0 && dist[t1+b]==dist[t1]-1))
     {
576   gira_sx(90);
      X[contatoreX]=2;
578   contatoreX++;
```

```
580     //Calcolo nuovo verso
        if (verso==0) verso=3;
582     else if (verso==1) verso=2;
        else if (verso==2) verso=0;
584     else if (verso==3) verso=1;
      }
586
      //Gira a destra
588   else
      if ((verso==1 && xr>0 && mappa[t1-1]==0 &&
590   dist[t1-1]==dist[t1]-1) ||(verso==0 && xr<b-1 &&
      mappa[t1+1]==0 && dist[t1+1]==dist[t1]-1) ||
592   (verso==3 && yr>0 && mappa[t1-b]==0 &&
      dist[t1-b]==dist[t1]-1) ||(verso==2 && yr<h-1 &&
594   mappa[t1+b]==0 && dist[t1+b]==dist[t1]-1))
      {
596    gira_dx(90);
       X[contatoreX]=3;
598    contatoreX++;

600    //Calcolo nuovo verso
       if (verso==0) verso=2;
602    else if (verso==1) verso=3;
       else if (verso==2) verso=1;
604    else if (verso==3) verso=0;
      }
606
      //Gira di 180 gradi
608   else
      {
610    gira_180();

612    //Calcolo nuovo verso
       if (verso<2) verso=1-verso;
614    else if (verso==3) verso=2;
       else if (verso==2) verso=3;
616    X[contatoreX]=4;
       contatoreX++;
618   }

620   //Controllo di non avere ostacoli davanti
      int dista=SensorUS(IN_4);
622   Wait(200);
```

```
      //Ostacolo trovato
624   if ((dista < DIST_MIN_OSTACOLO))
      {
626    ostacolo=1;

628    //Metto l'ostacolo nella mappa e rieseguo il wfa
       switch(verso)
630    {
        case 0:
632     mappa[t1-b]=1;
        break;
634
        case 1:
636     mappa[t1+b]=1;
        break;
638
        case 2:
640     mappa[t1+1]=1;
        break;
642
        case 3:
644     mappa[t1-1]=1;
        break;
646    }
       wave_front_algorithm();
648   }
      else
650   {
       ostacolo=0;
652    switch(verso)
       {
654     case 0:
                    yr--;
656     break;

658     case 1:
        yr++;
660     break;

662     case 2:
        xr++;
664     break;
```

```
666       case 3:
          xr--;
668       break;
         }
670
       if((dist[xr+yr*b]<1))
672    {
        ClearScreen();
674     TextOut(0,LCD_LINE4,"distMIN1 in orienta_correttamente()");
        Wait(3000);
676     wave_front_algorithm();
       }
678          }

680          //Riesegue finche' non ha strada libera davanti
    }
682  while (ostacolo==1);

684 }


686


688 sub muovi_robot()
    {
690  int tlight,t1;
     ArrayInit(X,0,area);
692
     //Usato per rilevare il superamento di una linea in rotazione
694  err=0;
     while ((arriv!=1))
696  {
      orienta_correttamente();
698   OnFwdReg(OUT_BC,POWER,OUT_REGMODE_SYNC);
      t1=0;
700   while (true)
      {
702    tlight = Sensor(IN_3);
       if ((tlight>SOGLIA_LUCE_MIN && tlight<SOGLIA_LUCE_MAX) && t1==0)
704    {
        //Entro in linea bianca
706     t1=1;
        PlayTone(440,100);
708          }
```

```
      else
710   if ((tlight<SOGLIA_LUCE_MIN || tlight>SOGLIA_LUCE_MAX) && t1==1)
      {
712    //Esco da linea bianca
       t1=0;
714    PlayTone(220,100);
       Off(OUT_BC);
716    break;
            }
718  }
     riallinea();
720  //Centramento nella casella
     avanza((LUNGHEZZA_CELLA-DISTANZA_RUOTE)/2);
722 }
    ClearScreen();
724 TextOut(0,LCD_LINE5,NumToStr(verso));
    Wait(8000);
726

}
728

//Main di beezer
730 task main()
{
732

 err=1;
734 crea_path();
 if (err==0)
736 {
  //Sensore sonar
738  SetSensorLowspeed(IN_4);

740  //Sensore di luce
  SetSensorLight(IN_3);
742

  muovi_robot();
744 }

746 }
```

## 6.2 Codice completo Robot Molly

```
1 /*
  * Codice Robot Molly
```

```
3   * Franco Marangoni & Zahir Fetovski
    *
5  */

7  /*
   azione
9  case 0: "Arrivato";
   case 1: avanza(400);
11 case 2: gira_sx(90);
   case 3: gira_dx(90);
13 case 4: gira_180();
   */
15
   #define MASTER    0    //Master channel is always 0
17 #define DISTANZA_RUOTE 110    //11.0 cm
   #define RAGGIO_RUOTE 16       //2.8 cm
19 #define POWER 80
   #define LUNGHEZZA_CELLA 400   //40.0 cm
21 //Distanza minima in cm a cui deve trovarsi l'ostacolo
   // per far cambiare percorso al robot
23 #define DIST_MIN_OSTACOLO 45
   #define GIRA 25
25 #define GIRA1 24
   #define SOGLIA_LUCE 53
27 #define SOGLIA_LUCE_MIN 50
   #define SOGLIA_LUCE_MAX 58
29 #define ASCII_ZERO 48
   #define VOL 7
31 #define T_MIN 650
   #define T_MAX 800
33 #define TAPPE 3

35 const byte camPort  =  IN_1;

37 /*
   mappa[] contiene la struttura della mappa:
39 mappa[ y*b + x] contiene la casella c[y][x] della griglia:
   mappa[ y*b + x]==1 --> cella con ostacolo
41 mappa[ y*b + x]==0 --> cella libera
   */
43
   int mappa[];
45 //dist[] contiene il calcolo di distanze dal goal ottenuto
```

```
//con il wave front algorithm

int dist[];
int err; //Segnalo a main se ho avuto errori in inizializzazione.
// x,y goal; x,y, orientazione robot; base, altezza, area griglia
int xg,yg,xr,yr,verso,area,b,h;

mutex moveMutex; //semaforo

byte __local_buffer[80];
byte __local_array[59];

// BLUETOOTH - ALLEGATO IN SEZIONE BLUETOOTH NXT


void blueget()
{
 byte fileHandle;
 short fileSize;
 short bytesWritten;
 string read;
 string write="";
 DeleteFile("movimenti.txt");
 CreateFile("movimenti.txt", 90, fileHandle);
 string r, m, tmp;
 int i = 0, j;
 //Inizializzazione NXT come slave
 slavecheck();
 WriteLnString(fileHandle,"5", bytesWritten);
 WriteLnString(fileHandle,"10", bytesWritten);
 Wait(1000);
 while((i<=11)&&(r!="00000"))
 {
  r = receivefrommaster();
  j = StrLen(r);
  if(j!=0)
  {
   TextOut(0, LCD_LINE1, NumToStr(i));
   TextOut(0, LCD_LINE5, "                ");
   TextOut(0, LCD_LINE5, r);
   Wait(2000);
   i++;
   WriteLnString(fileHandle,r, bytesWritten);
```

```
89    }
      Wait(NAP);
91    ResetSleepTimer();
     }
93   CloseFile(fileHandle);
    }
95
    //Avanza di dist mm
97  sub avanza(long dist)
    {
99
     long k;
101  Off(OUT_BC);
     k = dist*36000/(628*RAGGIO_RUOTE);
103  RotateMotorEx(OUT_BC, POWER, k, 0, true, true);

105 }

107 //Controllo di non avere ostacoli davanti
    int controllo(int ostacolo)
109 {

111  if (SensorUS(IN_4) < DIST_MIN_OSTACOLO)
     {
113   PlayTone(440,100);
      PlayTone(220,100);
115   ClearScreen();
      TextOut(0,LCD_LINE1,"Rilevato Ostacolo");
117   TextOut(0,LCD_LINE2,"Ricalcolo percorso...");
      ostacolo=1;
119  }
     else
121  {
      ostacolo=0;
123  }
     return ostacolo;
125
    }
127
    //Gira a dx di x gradi (fa perno su dx e muove sx)
129 sub gira_dx(long x)
    {
131
```

```
 long k;
 int i=TAPPE;
 err=0;   //Non ho passato la linea
 Off(OUT_BC);
 k =  x*GIRA/5;
 k/=TAPPE;
 while ( i-->0)
 {
  RotateMotor(OUT_C, POWER, -k);
  Off(OUT_BC);
  RotateMotor(OUT_B, POWER, k);
  Off(OUT_BC);
 }

}

//Gira a sx di x gradi (fa perno su sx e muove dx)
sub gira_sx(long x)
{

 long k;
 int i=TAPPE;
 err=0;
 Off(OUT_BC);
 k =  x*GIRA/5;
 k/=TAPPE;
 while ( i-->0)
 {
  RotateMotor(OUT_B, POWER, -k);
  Off(OUT_BC);
  RotateMotor(OUT_C, POWER, k);
  Off(OUT_BC);
 }

}

//Cambia il verso dell'orientazione
sub gira_180()
{

 long k;
 Off(OUT_BC);
 k =  180*GIRA1/5;
```

```
175  OnRev(OUT_B, POWER);
     OnFwd(OUT_C, POWER);
177  while(MotorTachoCount(OUT_C)<k){}
     Off(OUT_BC);
179  Wait(4000);

181 }

183 //Legge un numero da file
    int read_number(byte handle)
185 {

187  int i,ris=0;
     string s;
189  ReadLnString(handle, s);
     return StrToNum(s);
191
    }
193
    sub per_elisa()
195 {

197  PlayToneEx(659,250,VOL,FALSE); Wait(250);
     PlayToneEx(622,250,VOL,FALSE); Wait(250);
199  PlayToneEx(659,250,VOL,FALSE); Wait(250);
     PlayToneEx(622,250,VOL,FALSE); Wait(250);
201  PlayToneEx(659,250,VOL,FALSE); Wait(250);
     PlayToneEx(494,250,VOL,FALSE); Wait(250);
203  PlayToneEx(587,250,VOL,FALSE); Wait(250);
     PlayToneEx(523,250,VOL,FALSE); Wait(250);
205  PlayToneEx(440,750,VOL,FALSE); Wait(750);
     PlayToneEx(262,250,VOL,FALSE); Wait(250);
207  PlayToneEx(330,250,VOL,FALSE); Wait(250);
     PlayToneEx(440,250,VOL,FALSE); Wait(250);
209  PlayToneEx(494,750,VOL,FALSE); Wait(750);
     PlayToneEx(330,250,VOL,FALSE); Wait(250);
211  PlayToneEx(415,250,VOL,FALSE); Wait(250);
     PlayToneEx(494,250,VOL,FALSE); Wait(250);
213  PlayToneEx(523,750,VOL,FALSE); Wait(750);

215 }

217 //Smoke on the w Vedi codice Beezer
```

```
//Carica la mappa da file
sub crea_path()
{

 byte handle_in;
 int fsize;
 int i,j;
 string s, st1,st2,st3;     // stringhe temporanee
 int t,t2; //    int xg,yg,xr,yr,or,area
 //Esce se c'e' errore in apertura file, esci
 if(OpenFileRead("movimenti.txt", fsize, handle_in) != NO_ERR)
 {
  ClearScreen();
  TextOut(0,LCD_LINE1,"Impossibile aprire");
  TextOut(0,LCD_LINE2,"movimenti.txt");
  PlayTone(330,200);
  Wait(4000);
  return;
 }
 else
 err=0;
 b = read_number(handle_in);     //Carico base e altezza
 h = read_number(handle_in);
 area = b*h;

 //ArrayInit(dist,0,area);
 //Dist sar  compilata da wave_front_algorithm()
 //Inizializzo info: tutte libere tranne bordi
 ArrayInit(mappa,0,area);

 //Leggo il file e carico le info della mappa
 for (i=0; i<h; i++)
 {
  if (ReadLnString(handle_in, s) == NO_ERR)
  for (j=0; j<b; j++)
  {
   t = StrIndex(s, j);
   if (t==ASCII_ZERO+0)
   {
    //Colloco ostacolo in mappa
    mappa[i*b+j]=0;
   }
```

```
261     if (t==ASCII_ZERO+1)
        {
263      //Colloco ostacolo in mappa
         mappa[i*b+j]=1;
265     }
        if (t==ASCII_ZERO+2)
267     {
         //Colloco ostacolo in mappa
269      mappa[i*b+j]=2;
        }
271     if (t==ASCII_ZERO+3)
        {
273      //Colloco ostacolo in mappa
         mappa[i*b+j]=3;
275     }
        if (t==ASCII_ZERO+4)
277     {
         //Colloco ostacolo in mappa
279      mappa[i*b+j]=4;
        }
281    }
     }
283   CloseFile(handle_in);

285 }

287 sub muovi_robot()
    {
289
     int i,j, azione;
291  for (i=0; i<h; i++)
     {
293   for (j=0; j<b; j++)
      {
295    azione=mappa[i*b+j];
       switch(azione)
297    {
        case 0:
299     ClearScreen();
        avanza(200);
301     TextOut(0,LCD_LINE1,"Arrivato");
        Wait(1500);
303     j=b; i=h;
```

```
      break;
305
      case 1:
307   avanza(400);
      ClearScreen();
309   TextOut(0,LCD_LINE1,"AVANZA");
      Wait(1500);
311   break;

313   case 2:
      gira_sx(90);
315   ClearScreen();
      TextOut(0,LCD_LINE1,"SX");
317   Wait(1500);
      break;
319
      case 3:
321   gira_dx(90);
      ClearScreen();
323   TextOut(0,LCD_LINE1,"DX");
      Wait(1500);
325   break;

327   case 4:
      gira_180();
329   ClearScreen();
      TextOut(0,LCD_LINE1,"180");
331   Wait(1500);
      break;
333    }
     }
335  }
    }
337

339 task main()
    {
341
     blueget();
343  Wait(500);
     while(true)
345  {
      avanza(200);
```

```
347    gira_sx(90);
       Wait(200);
349    avanza(200);
       gira_dx(90);
351    Wait(2000);
     }
353  while(true)
     {
355    gira_180();
       Wait(2000);
357  }
     err=1;
359  crea_path();
     //Se ha calcolato il path muovi il robot
361  if (err==0)
     {
363    SetSensorLowspeed(IN_4);      //Sonar
       SetSensorLight(IN_3);         //Sensore di luce
365    muovi_robot();
     }
367
   }
```

## 6.3 Codice completo Robot Spike

```
   \*
2   * Codice Completo Robot SPIKE
    * Franco Marangoni & Zahir Fetovski
4   *
   */
6
   #define POWER 100
8  #define POWER_COLPO 40
   //Distanza minima in cm a cui deve trovarsi l'ostacolo
10 //o eventualmente il nemico per far cambiare percorso
   //al robot o per attaccare
12 #define DIST_MIN_OSTACOLO 10

14 sub avanza()
   {
16
    Off(OUT_BC);
18  RotateMotorEx(OUT_BC, POWER, 8000, 0, true, true);
```

```
20 }

22 //Controllo di non avere ostacoli o nemici davanti
   void controllo()
24 {

26  int dist=0;
    dist=SensorUS(IN_2);
28  if (dist<DIST_MIN_OSTACOLO)
    {
30   Wait(500);
     //Se vero = Nemico
32   if(SensorUS(IN_2)<dist)
     {
34    RotateMotor(OUT_A, POWER_COLPO, 40);
      Wait(1000);
36    RotateMotor(OUT_A, POWER_COLPO, -40);
      //Effettuato l'attacco, si gira e se ne va
38    gira_180(120);
      avanza();
40   }
     else
42   {
      //Random(100) ritorna un valore tra 0 e 100
44    direzione = Random(100);
      if(direzione > 50 )
46    {
       //Vista la struttura del robot con 120 come parametro
48     //Ruota di circa 90
       gira_dx(120);
50    }
      else
52    {
       gira_sx(120);
54    }
     }
56  }
    avanza();
58

   }
60

   //Gira a dx di x gradi (fa perno su dx e muove sx)
```

```
sub gira_dx(long x)
{

 RotateMotor(OUT_B, POWER, x);

}

//Gira a sx di x gradi (fa perno su sx e muove dx)
sub gira_sx(long x)
{

 RotateMotor(OUT_C, POWER, x);

}

//Cambia il verso dell'orientazione
sub gira_180(long x)
{

 Off(OUT_BC);
 RotateMotor(OUT_C, POWER, x);
 RotateMotor(OUT_B, POWER, -x);
 Wait(400);
 avanza();

}

task main()
{

 SetSensorLowspeed(IN_2);     // sonar
 while(true)
 {
  controllo();
 }

}
```

## 6.4 Materiale illustrativo realizzato per l'esposizione

L'esposizione di tutto il lavoro svolto per l'evento di Rovereto è stata completata con la realizzazione di materiale illustrativo al fine di facilitare la

comprensione del lavoro ad un pubblico molto vario. È stato dunque creato mediante Photoshop, un poster contenente una breve descrizione di tutti e tre i robot e alcuni accenni a dettagli e specifiche. Il poster in dimensione integrale è accessibile al seguente link :

`http://www.magiczuber.it/PosterDiscoveryOnFilmFetovskiMarangoni.jpg`

Inoltre é stata creata una presentazione video contenente i filmati delle singoli funzioni svolte dai robot affinché, in caso di eventuali imprevisti, si potesse gestirli non bloccando l'esposizione.

# Capitolo 7

# Conclusioni

## 7.1 Considerazioni finali sul progetto

La robotica è un settore in continua evoluzione e innovazione. A svilupparsi parallelamente con le nuove scoperte è l'educational robotics. L'educational robotics è un nuovo settore di ricerca che considera le tecnologie robotiche come oggetti con cui imparare. Sono stati avviati diversi laboratori sperimentali di robotica educativa in molte scuole ed università italiane, utilizzando i robot construction kits, come il kit Mindstorm citato sopra. Configurandosi come giocattoli ispezionabili, i robot permettono di interagire con meccanismi di costruzione e montaggio e di arrivare ad analizzare la programmazione del comportamento. Durante tale processo, la costruzione,ricostruzione del robot, nonché la programmazione del suo comportamento, secondo i continui feedback ricevuti dall'ambiente, consentono ai progettisti di esercitare ed incrementare varie complesse abilità cognitive. L'approccio a problemi di natura differente quali movimento, condizioni atmosferiche, algoritmi di ricerca, trasmissione dati e le relative soluzioni, realizzano quindi una forma mentis a 360 gradi. L'esperienza ha avuto un esito molto positivo proprio partendo da questi presupposti. Il proggetto è servito da stimolo per nuovi studi, nuovi approfondimenti, nuove soluzioni, stimolando fantasia ed ingegno.Un ulteriore arricchimento personale è arrivato grazie all'esposizione del progetto al Discovery On Film. Grazie all'iniziativa resa possibile dal Museo civico di Rovereto, abbiamo avuto l'opportunità di confrontarci con altre università, altri studenti, di ogni età e provenienza.

## 7.2 Ringraziamenti

Ringrazio innanzitutto il collega nonchè amico Marangoni Franco, per l'aiuto e sostegno non solo durante la realizzazione dell'intero progetto esposto in questa tesi, ma durante tutto il percorso di studi. Ringrazio la mia famiglia, che mi ha sempre appoggiato e sorretto nei momenti più difficili. Ringrazio il Professor Moro Michele, per l'infinita pazienza e disponibilità. Ma ringrazio soprattutto Cristina, che nel corso di quest'ultimo tratto del mio percorso universitario, mi ha permesso di affrontare ogni cosa più serenamente.

# Bibliografia


[1] Informazioni generali NXT, Driver, Tools di sviluppo : Sito ufficiale Lego Mindstorm - http://mindstorms.lego.com/

[2] Creating Cool Mindstorms Nxt Robots di Daniele Benedettelli

[3] Bluetooth NXT : Studi svolti da Sivan Toledo-Prof. della Blavatnik School of Computer Science Tel-Aviv University : http://www.tau.ac.il/ stoledo/lego/

[4] Caratteristiche AVRCam : Building robots instead of sleeping- http://www.jrobot.net/Projects/

[5] Manuale NXC : Next Byte Codes e Not eXactly C http://bricxcc.sourceforge.net/nbc/

[6] Corso NXC - http://didattica.fuss.bz.it/Robotica

[7] Teoria Bluetooth : The official bluetooth Technology- http://www.bluetooth.com/