UNIVERSITÀ DEGLI SUDI DI PADOVA

FACOLTÀ DI INGEGNERIA

DIPARTIMENTO DI INGEGNERIA INFORMATICA

Tesi di laurea

Piattaforma di sviluppo per il coordinamento di behavior su robot a minima sensoristica

Realtore: Prof. Enrico Pagello

Laureando: Maragno Simone

ANNO ACCADEMICO 1999-2000

Alle donne della mia vita,
mia madre
e la Chiara

Non importa quanto tu possa primeggiare
nell'arte della mano vuota
e nei tuoi tentativi scolastici.
Nulla è più importante del tuo comportamento
e della tua umanità
visti nella vita di tutti i giorni

Nagamine sensei

Ringraziamenti

Ringrazio il Professore Enrico Pagello che mi ha dato l'opportunità di svolgere questa tesi, ringrazio tutte le persone che mi sono state vicino in questo periodo, la mia ragazza, senza la quale non vivrei, i miei famigliari per aver sopportato e mantenuto i miei studi in questi anni, il Karatedo, tutta la palestra SKS Padova dove pratico ed insegno, il maestro Tessarolo che ci segue da anni con passione.

Indice

Capitolo 1 Coordinamento di behavior su robot a minima sensoristica

Coordinamento di behavior su robot a minima sensoristica

1.1 Introduzione

Nel paradigma *behavior-based* [Arkin 98, Matarik 97], il controllo di un robot è diviso tra un set di processi precostituiti , chiamati *behaviors* (comportamenti). Data un'informazione (selezionata da quelle provenienti dai sensori) ogni *behavior* produce e/o propone una reazione per controllare il robot rispetto ad un particolare obbiettivo come il completamento di una parte di un *task.*

Behaviors con differenti obiettivi possono produrre azioni in conflitto tra loro apparentemente non riconducibili ad alcun tipo di interazione tra essi voluta . Quindi, lo sviluppo e lo studio dei meccanismi di *behavior coordination* e' a tutt'oggi tra i temi maggiori di ricerca.

Questa tesi si occupa di porre le basi per lo studio di tutti i tipi di coordinamento possibili su robot dotati di limitate capacità di elaborazione e quindi dotati di "semplici" sensori , ad esempio una microcamera è già un sensore complesso che richiede grosse capacità di calcolo.

Piattaforma di sviluppo significa studiare, data una struttura di base, quali sono le migliori architetture meccaniche, software ed hardware necessarie per implementare al meglio i coordinamenti desiderati

Lo scopo finale non è, dato un task sviluppare le modifiche opportune per raggiungerlo al meglio, ma creare un struttura indipendente dal singolo task in cui è possibile applicare il maggior numero di azioni coordinate .

1.2 Robot a minima sensoristica

L'accezione di robot qui usata ha il significato di *Intelligent Autonomous System* dove la parola System può essere sostituita con *Agent*

1.2.1 Agent

Definizione di Agent

[Steel 94b, 94c]

Un Agente è un sistema : Ciò significa è composto da un set di elementi che possiedono una relazione tra loro e con l'ambiente. Non devono essere necessariamente fisici.

Un agente svolge una certa funzione per qualcun altro : il significato è intuitivo, facendo riferimento alla parola agente, che agisce e compie una qualche azione.

Un agente è un sistema in grado di auto-sostenersi : Un agente deve adempiere ad un a sua funzione, ma nel contempo deve mantenersi in grado di adempiere la sua funzione , quindi "sopravvivere"

Tale definizione quindi è molto simile a quella che si dà in genere per un essere vivente. In particolare sono due i requisiti che si richiedono ad un agente per essere paragonabile in questi termini ad un essere viventi:

- Il sistema, nella sua interezza, si adatta/evolve per rimanere "vivo" anche a fronte di cambiamenti ambientali.
- I sistemi viventi sostituiscono in continuazione in loro componenti, per assicurarsi efficienza.

In definitiva si deve osservare che senza la prospettiva biologica, è estremamente difficile distinguere tra agenti ed altri tipi di sistemi software o hardware. La spinta alla auto conservazione nelle creature viventi si trova in biologia a più livelli. Equivalenti livelli sono individuabili per agenti robotici

- Livello genetico: attraverso Algoritmi genetici nella cosiddetta *Evolutionary robotics*
- Livello strutturale: nella connessione tra stati sensoriali e attuativi
- Livello di gruppo: è il caso dei *multi robots systems*

In definitiva per robot si è inteso un oggetto fisico, soggetto alle leggi descritte dalla fisica, che ha la capacità di controllare il proprio destino. Un sasso per esempio non può controllare i propri movimenti, è semplicemente soggetto alla forza di gravità.
In questa tesi verrà studiato l'oggetto robot nella sua rappresentazione meccanica, hardware e software.

1.2.2 Autonomy

Definizione di Autonomy

[Smiters 92] Autonomia deriva dal greco: *autos* (se stesso) e *nomos* (regola o legge). Ossia chi si dà le regole da sé. In contrasto con Automatico, che deriva dal greco autogoverno nel senso che si regola o calibra da sé, ma che non ha generato e non genera le regole che utilizza per calibrarsi o regolarsi: le regole sono immesse dall'esterno.

McFarland [Mcfarland 93] afferma che un sistema automatico è un sistema del quale si è in grado di predire il comportamento, (utilizzando il controllo classico), avendo a disposizione le regole che utilizza. Un sistema autonomo invece non permette questa determinatezza, lasciando solo una idea di quello che sa accadendo all'interno del sistema Ciò fa trasparire l'importanza di studiare il coordinare di tali comportamenti, come detto da Smithers [Smithers 92] il problema dei sistemi autonomi è capire come questi sviluppino e modifichino i principi attraverso i quali regolano i loro *behaviors* continuando a ricercare il raggiungimento del loro obiettivo .

I Sistemi Autonomi sono quindi sistemi che sono in grado di sviluppare da soli , al loro interno le regole che poi utilizzeranno per regolare i loro *behavior* . Quindi Autonomia implica Automaticità ovvero per essere autonomo deve prima essere automatico (e la gran parte della robotica conosciuta nelle industrie lo è)

1.2.3 Intelligent

Definizione di Intelligent

[Steel 94c]
Confronto di prestazioni con umani: È la definizione storica, usata nel test di Turing. È basata sul confronto delle prestazioni e trova ad esempio nel gioco degli scacchi la sua de-

bolezza. Non cattura l'idea, ma solo una parte superficiale degli aspetti coinvolti dell'intelligenza.
Conoscenza ed intenzionalità: Si basa sulla rappresentazione della conoscenza. Newell ha descritto il knowledge level. Tale descrizione può essere fatta su un sistema sul cui comportamento si ragiona in termini di conoscenza posseduta e applicazione di tale conoscenza, secondo il principio di massima razionalità. Un sistema è definito intelligente se è possibile costruire il suo knowledge level e se massimizza l'uso di tale conoscenza in una data situazione, conosce un certo numero di cose e fa la scelta giusta. Pare quindi che l'Intelligenza Artificiale (AI) stia tutta nell'estrazione e formalizzazione della conoscenza.
Tale definizione ha due limiti:

- Si può applicare a sistemi che normalmente non vengono considerati intelligenti (termostato)
- Produce una netta separazione tra sistemi intelligenti e non , rendendo difficile spiegare come può essere nata l'intelligenza in sistemi fisici (neurofisiologia).

Comunemente in letteratura la nozione di intelligenza si manifesta in due abilità:

- *Action-centered*: queste abilità si manifestano nelle attività sensomotorie come camminare attorno ad una casa, guidare una macchina, suonare un piano.
- Intellettive: queste abilità sono necessarie per sviluppare ed implementare programmi di computer, gioco degli scacchi o formule matematiche.

Il professor Zuboff [Zuboff 88] la distingue in quattro caratteristiche:

- *Sentience*: L'abilità *action-sentience* è basata sulla sensazione derivata da stimoli fisici. Le abilità intellettive sono basate su un'astrazione espressa come descrizione.
  Essa può quindi operare in assenza di diretti stimoli fisici.
- *Action-dependence*: La prima abilità si sviluppa in performance fisiche. Sebbene in principio essa possa essere esplicitata in un linguaggio, tipicamente rimane inespressa: è il caso di un'azione implicita. Lo sviluppo di attività intellettive invece, si basa su elaborazioni compiute durante lo svolgimento dell'azione o attraverso comunicazioni di tipo verbale oppure scritto.
  L'action-dependence può essere quindi esplicitata o realizzata in modo esplicito.
- *Context-dependence*: L'abilità *action-sentience* ha significato solo in un contesto in cui l'attività fisica ad essa associata può realizzarsi. La seconda abilità è *contexts-independence* perché lavora su basi astratte.
  Il collegamento tra le descrizioni e le abilità necessita di essere stabilito in maniera esplicita.
- *Personalism*: È data dal manifestarsi delle competenze richieste nelle azioni individuali. Esiste un rilevante collegamento tra il conoscente e il conosciuto. Le abilità intellettive sono basate su conoscenze sganciate dal corpo "fisico" che molto spesso sono di tipo distribuito. La *personalism* è molto spesso data dalla cultura di un individuo, trasmessa attraverso libri o insegnata esplicitamente attraverso l'educazione.

Ovviamente la ricerca relativa alle attività intellettive fa parte dell'AI**,** intesa come branca della robotica, ed esula dal contenuto di questa tesi.
Nel nostro robot l'intelligenza sarà data dal cosiddetto comportamento emergente che può essere solamente determinata dal totale comportamento del sistema e da come tale comportamento appare in relazione con l'ambiente che lo circonda. L'idea rispecchia quanto affermava Brooks [Brooks 91]:

"Intelligence is in the eye of the observer"

L'intelligenza sta negli occhi dell'osservatore, ed ancora una volta mette in risalto quanto sia fondamentale determinare il comportamento, (behavior), del robot, questo è possibile se e solo se si sono studiati gli effetti del coordinamento di behavior più elementari.

1.3 Lego RCX

1.3.1 Struttura ideale

Per la realizzazione del robot[1] si è cercata una struttura che rispondesse ad esigenze di:

- Basso costo: Per rispettare le specifiche di "minima sensoristica"; infatti sarebbe stato incongruente lavorare con una struttura potente/complessa ,(dal punto di vista computazionale) e quindi con un costo elevato ed istallargli sensori semplici: non avremmo sfruttato la tecnologia a nostra disposizione con un conseguente spreco di risorse per la ricerca .
- Manutenibilità: È una caratteristica necessaria in un contesto di progetto controllo e revisione del robot essa si divide in 3 sottoclassi
  - Correggibilità: Capacità di poter modificare la struttura del robot con il minor impiego di tempo e risorse, senza danneggiarlo , al fine di correggere/migliorare eventuali difetti o carenze
  - Testabilità : La struttura deve permettere il monitoraggio delle variabili di controllo (preventivamente definite), questo può avvenire tramite visione diretta, misurazione strumentale od acquisizione software.
  - Espandibilità: Capacità di poter modificare la struttura del robot con il minor impiego di risorse e minor danno , al fine di espandere una o più delle sue caratteristiche queste possono riguardare la parte meccanica, i sensori, o il software in dotazione

  L'ultima sottoclasse non può essere applicata in modo iterativo ovvero non è possibile espandere il robot all'infinito esiste un Upper-bound di tipo fisico (meccanico), uno hardware ed uno software,se uno dei tre viene superato tutta la "struttura robot " ne risentirà. Inoltre spingersi troppo oltre ,con modifiche ed espansioni, metterebbe in luce la necessità di dover utilizzare una struttura molto più complessa di quella di partenza e conseguentemente rivelerebbe una errata pianificazione di sviluppo del robot, avvenuta prima dell'acquisto, in cui sarebbe strato più opportuno munirsi da subito di una struttura più complessa senza perdere tempo , risorse denaro.
- Modularità: È la capacità di poter dividere il robot in "zone" che possono essere progettate in modo indipendente ed alla fine assemblate vale per :
  - Meccanica: Zona motrice, zona sensore A, sensore B…
  - Software : Task (zona), per gestire motori, task per i singoli behavior, task di controllo, task gestione dei tasti..
- Portabilità:Se stessimo parlando di applicazioni puramente sosftware indicherebbe la capacità di del sorgente di essere "portato" ,fatto girare, su diverse macchine e/o sistemi operativi. Nel nostro caso si intende la capacità di poter essere usato, modificato, testato etc. da persone diverse da quelle che hanno sviluppato l'originale: la conoscenza necessaria per poter operare con il robot non deve essere di un singolo soggetto (senza il quale nulla va avanti) ma di tutti e quindi i mezzi con cui si può raggiungere questa conoscenza devono essere facilmente reperibili e ben documentati.

[1] Inteso come struttura fisica (meccanica), hardware e software

La struttura che meglio risponde a queste esigenze è il LEGO© MINDSTORM™ ROBOTICS INVENTIN SYSTEM 1,5 . Trattandosi di Lego il robot può essere facilmente modificabile, togliendo dei mattoncini e mettendone altri a nostro piacere (anche da altre scatole di LEGO TECHNIC)… conosciamo le loro potenzialità : fin da piccoli li abbiamo usati[2]

1.3.2 LEGO MINDSTROM Robotic invention system

Nella scatola sono presenti oltre 700 pezzi (non solo mattoncini, vedasi appendice),2 motori,2 bump sensor (sensore di contatto), 1 light sensor (sensore di luminosità), 1 IR trasmitter (trasmettitore infrarosso) da collegare al computer ed il cuore di quello che sarà il futuro robot: l'RCX (Figura 1.1).
Vi è inoltre un manuale, con pochi ma chiari esempi costruttivi ed un Cdrom grazie al quale è possibile scaricare il firmware,(il sistema operativo), ed un semplice linguaggio di programmazione ad icone per programmare l'RCX .
Non comprese nella scatola sono necessarie 6 batterie da 1.5 volt per il brick (termine con cui si indica l'RCX), ed 1 da 9 volt per IR trasmitter

RCX:(acronimo di Robotic Command System), è un microprocessore LEGO autonomo che può essere programmato utilizzando un PC. L'RCX funge da cervello delle invenzioni di LEGO MINDSTORMS. Utilizza sensori per prelevare input dal proprio ambiente, elaborare dati e segnalare ai motori di uscita di accendersi o spegnersi.
Per far funzionare il robot da noi progettato bisognerà innanzitutto costruirlo utilizzando l'RCX e i pezzi LEGO. Successivamente, potrà creare un programma per la propria invenzione utilizzando RCX Code .Infine, scaricare il programma nell'RCX utilizzando il trasmettitore a raggi infrarossi. Ora il robot potrà interagire con l'ambiente, in tutta autonomia dal computer.

L'RCX dispone di :

- 3 porte per sensori di Ingresso (1,2,3)
- 3 porte d'uscita (A,B,C) per i motori
- 4 tasti di controllo
- 1 display LCD
- 1 trasmettitore a raggi infrarossi

È inoltre dotato di un microprocessore interno per l'elaborazione dei programmi , una memoria per la memorizzazione del firmware e programmi ed un altoparlante incorporato per emetter suoni e tonalità diverse. Per ulteriori dettagli si rimanda al paragrafo 2.2

Essendo un mattoncino LEGO, dispone di fori e bottoncini a incastroche consentono di collegarlo ad altri mattoncini LEGO

Porte sensore: sono utilizzate per collegare i sensori ottici e di contatto, così come i sensori di rotazione e temperatura non inclusi nel set base.

Porte di uscita: sono utilizzate per collegare i motori, così come le luci ed altre periferiche di output, non incluse nel set base

Tasti: consentono di controllare l'RCX e i relativi programmi (Figura 1.2)
On/Off (ROSSO): permette di accendere e spegnere il brick. Gli altri tasi funzionno solo se l'RCX è acceso.

[2] Almeno quelli della nostra generazione, cresciuti senza playstation e gameboy

Prgm (BIANCO): significa Program,consente di utilizzare i 5 alloggiamenti di programma dell'RCX. Il numero del programma selezionato viene visualizzato sulla destra "dell'omino" che compare sul display
Run (VERDE): avvia ed interrompe il programma selezionato. In modalità "Run" l'omino viene visualizzato in movimento
Viev (NERO) :attivo solo dopo il download del firmware , consente di ottenere informazioni sui sensori e i motori. I valori relativi ai sensori vengono visualizzati sul display in corrispondenza delle porte di ingresso (sotto ai numeri 1,2,3) e la direzione dei motori in corrispondenza delle porte di uscita (sopra le lettere A,B,C)

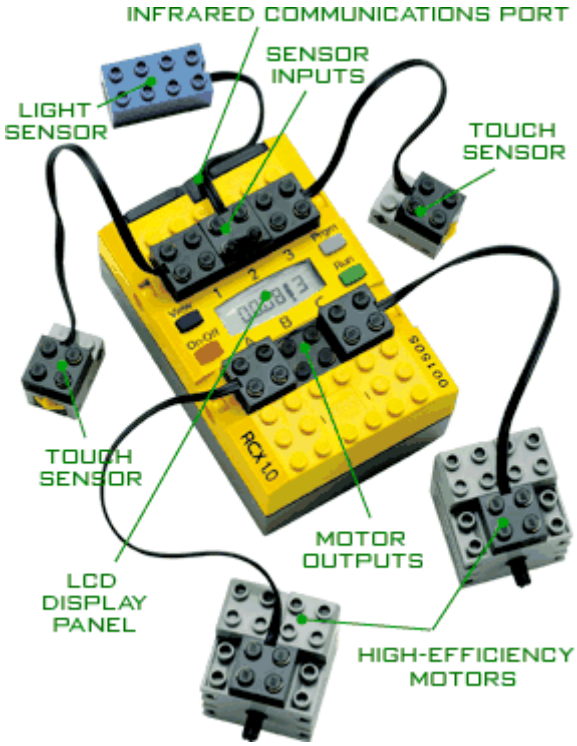


Figura 1.1 RCX con sensori e motori

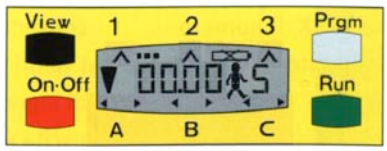


Figura 1.2 Display e tasti

Display: Visualizza le informazioni relative all'RCX ed ai programmi, in Figura 1.2sono presenti tutte le possibili visualizzazioni del display.

- ESECUZIONE PROGRAMMA:L'omino (che in Figura 1.1 sta in piedi piedi), comincia a muoversi quando viene premuto il tasto Run, quindi omino in movimento indica l'esecuzione di un programma nell'RCX .
- NUMERO PROGRAMMA: Il numero che compare a destra dell'omino può essere compreso tra 1 e 5 e indica quale dei 5 programmi verrà attivato alla pressione del tasto Run
- STATO DELLE BATTERIE:Lo stato di carica delle batterie viene visualizzato dal relativo indicatore. Quando le batterie sono quasi scariche, l'indicatore lampeggia e l'RCX emette un suono e si trova tra l'indicatore di sensore 2 e 3.

- TRASMISSIONE IR: Durante la comunicazione tra il Trasmettitore RI e l'RCX viene visualizzato un cono che indica il tipo di intervallo, di breve o lunga durata, della comunicazione a raggi infrarossi.
- DOWNLOAD DA PC: Se il download di un programma viene eseguito dal PC sull'RCX, viene visualizzata una sequenza di puntini,(in figura sembrano 3 quadratini).
- OROLOGIO: L'orologio, attivo solo dopo il download del firmware, indica da quanti minuti è acceso l'RCX dopo l'ultimo azzeramento. Accendendo, spegnendo l'RCX o scaricando di nuovo il firmware, l'orologio viene azzerato.
- SENSORI:Se sotto una porta sensore viene visualizzata una freccia, tale porta è utilizzata dal tasto View.
- USCITE: Se sopra una porta di uscita viene visualizzata una freccia, tale porta è attiva. La direzione della freccia indica la direzione di un motore collegato alla porta.
- DATALOG: in figura non è presente , viene visualizzato a destra del numero del programma, come un cerchietto diviso in quattro parti: le parti indicano quanta percentuale di datalog stiamo utilizzando. Il datalog è un array interno in cui memorizziamo dati. (per dettagli vedasi Par. 2.3 Firmware)

Capitolo 2 Struttura di base

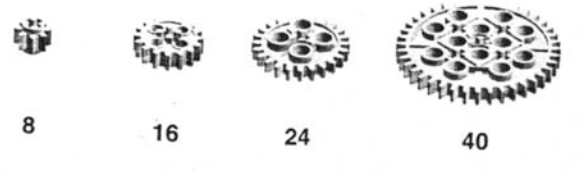
Per struttura di base si intende tutto il materiale fornito dalla lego che concorre nel realizzare il robot, quindi avremo una struttura meccanica, un hardware (l'interno dell'RCX), un firmware (il sistema operativo), un software per la programmazione .

2.1 Meccanica di base

La meccanica di base con cui è stato costruito il robot è strettamente legata ai pezzi LEGO di cui si dispone, oltre ai soliti mattoncini rettangolari od a forma di trave più o meno lunghi, dotati di fori per permettere l'inserimento e/o l'incastro di assi di varia lunghezza.

2.1.1 Gears

I motori in un certo senso mantengono vivo il robot perché gli permettono di spostarsi e quindi di interagire con l'ambiente ma in molti casi essi sono in posizioni scomode, ruotati rispetto all'asse di marcia o producono una velocità insoddisfacente. Per ovviare a tali problemi si introducono delle *gear tooth* (ruote dentate), **Tabella 2.1**



Disponibili	Denti	Raggio
6	8	0.5
4	16	1.0
4	24	1.5
4	40	2.5

Tabella 2.1 Basic gear

Gear reduction: (rapporto di riduzione) con questo termine viene indicata la riduzione del numero di rivoluzioni che si ottiene collegando tra motore e carico più ruote dentate di diametro diverso.

In meccanica viene spesso applicato il cosiddetto motoriduttore[3], esso consente infatti il pilotaggio di un carico ad elevata inerzia mediante una coppia motrice altrimenti insufficiente.

Il rapporto di riduzione viene calcolato mediante un "rapporto" tra il numero dei denti delle ruote comprese tra motore e carico,vedasi Tabella 2.2

Denti	8	16	24	40
8	1:1	1:2	1:3	1:5
16	2:1	1:1	1:2	2:5
24	3:1	2:1	1:1	4:5
40	5:1	5:2	5:4	1:1

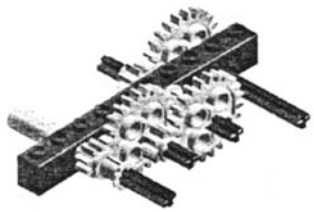


Tabella 2.2 Gear reduction ed esempio gear train

Per esempio collegando le ruota più piccola con la più grande si ottiene un gear reduction di 5:1 (40/8=5) ovvero sono necessari 5 giri della prima ruota per farne compiere 1 alla seconda quindi la seconda ruota è 5 volte più lenta dell'altra.

Vi può essere anche un incremento di giri ma ciò è determinato da quale ruota noi consideriamo collegata al motore e quale al carico; riprendendo l'esempio se poniamo il motore

[3] Dispositivo dotato di 2 ruote dentate di diametro diverso

nella ruota da 40 denti avremo un *gearing up* di 1:5 basta un giro della ruota grande per farne compiere 5 alla piccola.
Sono possibili anche multiple combinazioni di ruote (*gear train)* in tali catene il gear ratio di ogni coppia è moltiplicato a quello delle altre. Per esempio
Crown gears: È una ruota di 24 denti che permette il collegamento con altre ruote il cui asse di rotazione non è parallelo quello della crown gear ma perpendicolare; ovvero è possibile collegare due ruote a novanta gradi tra loro se almeno una è crown gear. Questo perché i suoi denti sono curvati è comunque importante che la ruota abbia ampio supporto direttamente dietro al punto di ingranamento altrimenti tenderà a scivolare

Figura 2.1 crown gears

Figura 2.2 bevel gear

Bevel gears:È una ruota da 12 denti molto compatta e si usa in alternativa alla crown gear. La sua struttura a "tronco di cono" la rende collegabile solo ad altre belvel gears questo svantaggio viene compensato dalla sua proprietà di trasferire la potenza molto più efficacemente della crown gear. Essendo piccole per un efficace utilizzo è consigliabile posizionarle su basi solide come le travi

Worm gear: È unica. Ogni sua rotazione corrisponde al movimento di un dente della ruota ad essa connessa, provvede quindi ad un estremo e compatto meccanismo di gear reduction: fattore riduzione è n:1 dove per "n" si intende il numero di denti dela ruota a cui è connessa. Al contrario delle altre ruote non ha denti ma un filetto che percorre tutta la sua struttura cilindrica inoltre è un dispositivo *one-way* ovvero la sua rotazione fa girare la ruota ad essa connessa se la worm gear è posta con asse di rotazione verticale (Figura 2.3), ma il viceversa non vale: il trasferimento di energia meccanica è unidirezionale. Se invece si cambia la posizione dell'asse di rotazione potandolo orizzontalmente (figura girata di 90 gradi in senso orario), sarà la rotazione della ruota ad essa connessa a permettere il trasferimento del moto

Figura 2.3 worm gear (in verticale) **Figura 2.4** rack (tra 2 lego) **Figura 2.5** differenziale

The Rack: La rack può essere usata per convertire la rotazione di una ruota in movimento lineare e viene solitamente utilizzata una ruota da 8 denti. Potrebbero anche essere utilizzare ruote più larghe, ma nella maggior parte delle applicazioni il controllo più preciso dato dalle ruote piccole risulta la scelta migliore. La rack deve essere in grado di scivolare liberamente indietro e in avanti, per tale motivo è opportuno che la base sia posta su una superficie liscia e non su un mattoncino lego classico.

Differential: Il differenziale è la ruota assemblata più complessa, ma è anche una delle più utili. Internamente utilizza 3 ruote tipo bevel per collegare le due assi indipendenti con

il corpo del differenziale. Le assi possono ruotare a differenti velocità, ma la loro velocità media deve uguagliare quella del differenziale. La bevel gear di mezzo (quella non attaccata ad alcun asse) deve essere posizionata all'interno del differenziale per prima. In seguito vengono inserite le altre due bevel gear ed infine i due assi corrispondenti.
Se si fa compiere al corpo del differenziale una rotazione completa, anche i due assi compieranno una rotazione completa, se lasciati liberi di ruotare, quindi tutto gira alla stessa velocità. Se invece si fanno ruotare gli assi alla stessa velocità, in direzioni opposte, il differenziale rimarrà fermo, poiché la velocità media dei due assi indipendenti è zero.

2.1.2 Pulleys

Come nel caso delle ruote dentate le pulegge (*pulleys*) possono essere usate per convertire velocità e potenza, ma le similitudini si concludono qui. Mentre le ruote invertono di direzione quando sono collegate ad un'alta, le pulleys girano nella stessa direzione; non avendo denti il rapporto tra pulegge è determinato unicamente dalla loro size (Tabella 2.3), sebbene non bisognerebbe mai fare affidamento unicamente su pulegge se si necessita di un rapporto esatto a causa del probabile slittamento

Pulley	Relative size
Small	2
Friction	3
Medium	7
large	11

Tabella 2.3 Pulley

Belts: Le pulegge non si connettono tra loro direttamente;la potenza viene trasferita attraverso cinghie (*belts*) di gomma, ciò significa che la distanza tra queste può essere molto flessibile. Il loro utilizzo è richiesto dove l'impiego di ruote dentate non è realizzabile. Lo spessore delle belts determina quanta può essere trasferita prima che la cinghia inizzi a scivolare.
In un primo momento tale scivolamento potrebbe sembrare una limitazione, invece è considerata una importante caratteristica delle pulegge in quanto provvede ad una sorta di "valvola di sicurezza" in un sistema dive un motore è attivo spesso con limitati range di moto.

2.1.3 Struttura a tank

La prima struttura implementata (Figura 2.6) è un modello tipo tank (cingolato). Si utilizzano alcune travi per fissare l'RCX alla base della struttura, il moto è trasmesso da motori posti posteriormente al brick, attraverso una catena di 2 ruote da 24 denti (rapporto di trasmissione 1:1).

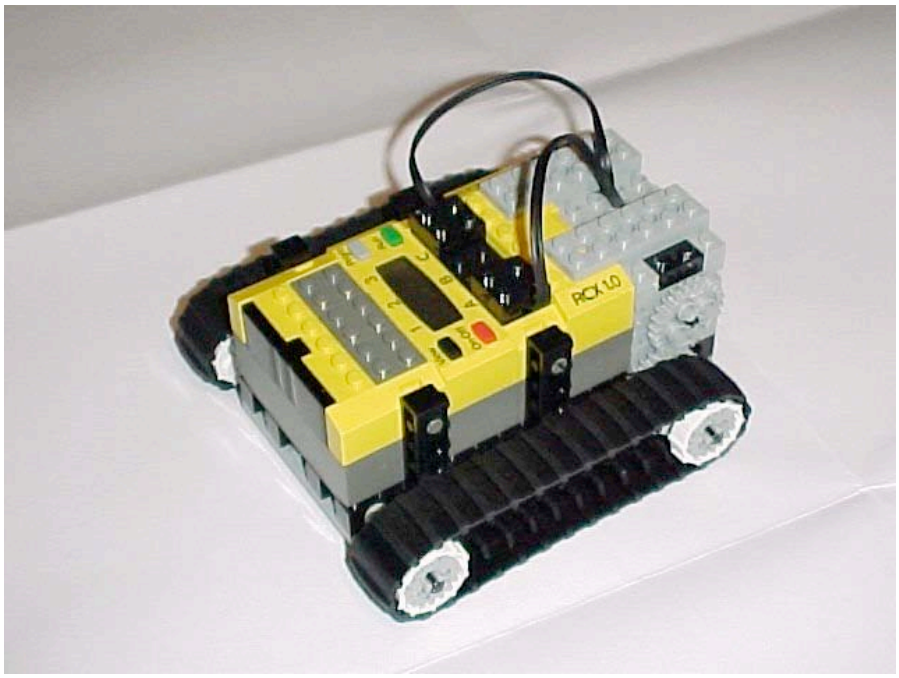


Figura 2.6: **Struttura tipo tank**

Il modello tipo tank (cingolato) [Borenstein 96] è una speciale implementazione di differential drive[4] conosciuta come skid steering ed è normalmente utilizzata in veicoli come bulldozer e blindati. Tale configurazione skid-steer fa affidamento sulla normale operazione di slittamento delle ruote , quindi opera con una determinazione del punto stimato insufficiente. L'effettivo punto di contatto di tali veicoli è irregolarmente vincolato sul bordo di una zona rettangolare di ambiguità, corrispondente all'orma della traccia del cingolo. In Figura 2.7, dai cerchi concentrici, si nota anche come possa avvenire un considerevole slittamento in corrispondenza di una curva [Everett 95].
Per tali ragioni questi veicoli sono generalmente impiegati nella robotica industriale su terreni sconnessi e discontinui dove non è richiesta un'accurata autolocalizzazione spaziale, e guidati via radio.

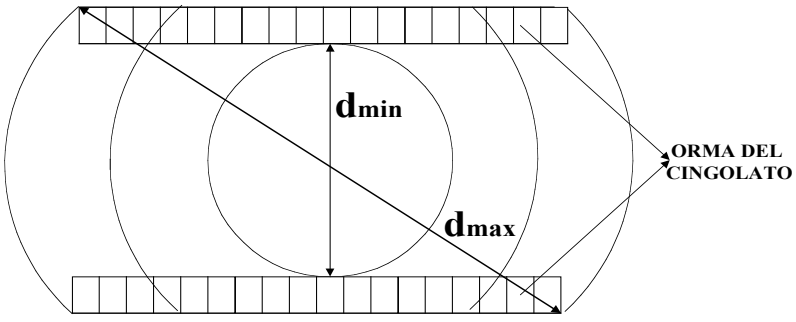


Figura 2.7 slittamento ruote

[4] Moto in cui le ruote sono alimentate motori distinti.

2.2 Hardware di base:

Il cuore del RCX (la CPU) è un microcontrollore Hitachi H8 con 32k di RAM esterna. Il microcontrollore è usato per il controllo dei 3 motori, 3 sensori, uno speaker e la porta seriale infrarossa. Vi è anche una ROM da 16k on-chip contenente un driver che si attiva all'accensione del RCX.
Per accedere a tutti questi componenti è necessario per prima cosa aprire l'RCX

2.2.1 Apertura dell'RCX

AVVERTENZA:Tutte le informazioni che seguiranno sono a scopo puramente conoscitivo, tutte le modifiche fisiche che il lettore produrrà al suo RCX sono a proprio rischio, in quanto qualsiasi azione non accurata può provocare la rottura del dispositivo

Per rimuover la circuiteria di bordo [Proudfoot 99],una vollta aperto l'RCX è necessario svitare le 4 viti, rimuovere il materiale plastico in cui è avvolto il trasmettitore infrarosso ,separare il coperchio superiore dell'RCX dal circuito di bordo ed infine sganciare i 2 contatti delle batterie: finalmente il circuito è libero[5].
Il contatto negativo delle batterie è piuttosto semplice da liberare rispetto a quello positivo. Infatti rimovendolo per primo si riesce a far scivolare fuori il circuito mentre per l'altro è necessario l'inserimemento di un cacciavite tra la slot e l'involucro del contenitore delle batterie e sforzare delicatamente verso fuori e verso l'alto.

In Figura 2.8 si possono notare da sinistra a destra, partendo dall'alto:

- Il circuito interno
- Il coperchio superiore del RCX
- Il copri trasmettitore infrarosso
- Le 4 viti
- I contatti delle batterie
- Il contenitore delle batterie
- Il coperchio inferiore del RCX

Figura 2.8 RCX smontato

[5] È anche possibile girare il circuito liberamente senza rimuovere il contatto delle batterie (Patrick Gili report)

Nella parte superiore del circuito (Figura 2.9) sono visibili:

- 2 LED infrarossi
- 1 ricevitore infrarosso
- 1 LCD al centro
- svariati condensatori

L'LCD copre un LCD controller e lo speaker; sopra e sotto l'LCD sono presenti i contatti per i 4 tasti di gomma. Infine 12 morsetti escono dal circuito per accoppiarsi con le porte di input e output.

Sul fondo del RCX ,Fig 2.10, si nota un largo quadrato: si tratta della CPU: il microcontrollore Hitachi. Più in basso si scorge un altro chip: la RAM , quello grande in mezzo invece è un banco di flip-flop, infine sopra vi è un piccolo banco di porte NAND.
Si suppone [Phillips 99] che i 3 identici chip in centro a sinistra servano per il controllo dei motori

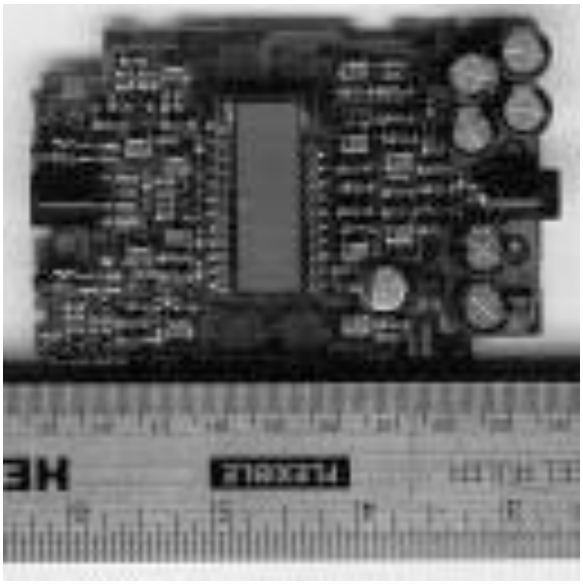


Figura 2.9 Lato superiore

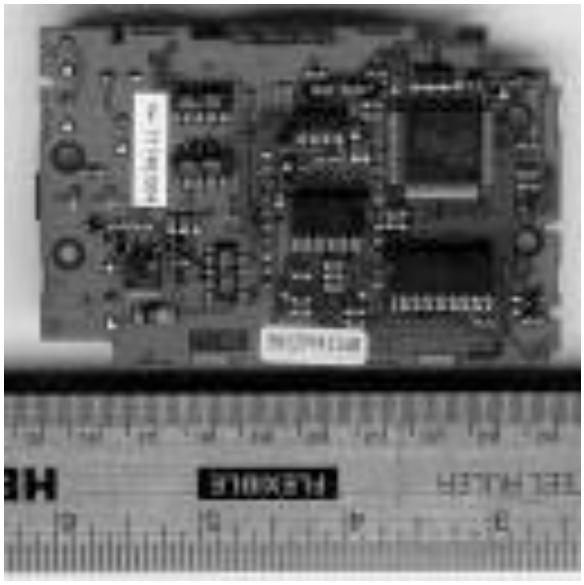


Figura 2.10 Lato inferiore

2.2.2 CPU

[Nielsson 00] Il brick RCX utilizza un microcontrollore *Hitachi* H8/3292, della famiglia H8/300L:

Product name	H8/3292
ROM size	16K
RAM size	512
Speed	16MHz a 5V
8-bit Timers	2
16-bit Timers	1
A/D Conversion	8 8-bit
I/O pins	43
Input only pins	8
Serial port	1
10mA outputs	10

Questo microcontroller è dotato di un set di istruzioni ottimizzato, adatto ad applicazioni in cui vi è la necessità di elevate velocità di calcolo. Supporta un indirizzamento a 16 bit ,

dispone di 16 registri da 8 bit (R0H, R0L, …, R7H, R7L), che possono essere usati anche come 8 registri da 16 bit (R0, …, R7) per scopi di indirizzamento. Ci sono due control registers: un program counter register (PC) a 16 bit e un conditions code register (CCR) a 8 bit. Il registro R7 è usato come stack pointer register e punta alla sommità dello stack (lowest address). Nel linguaggio assembly SP è sinonimo di R7. Tutte le operazioni degli stack avvengono con 2 byte words.
In Tabella 2.4sono mostrate tutte le caratteristiche degli H8/300L. [Hitachi 1]

General register configuration
- 16 8-bit registers (can be used as 8 16-bit registers)

55 basic instructions
- Multiply and divide instructions
- Powerful bit manipulation instructions

8 addressing modes
- Register direct (Rn)
- Register indirect (@Rn)
- Register indirect with displacement (@(d: 16, Rn))
- Register indirect with post-increment/pre-decrement
- (@Rn+/@-Rn)
- Absolute address (@aa:8/@aa:16)
- Immediate (#xx:8/#xx:16)
- Program-counter relative (@(d:8, PC))
- Memory indirect (@@aa:8)

64-kbyte address space

High-speed operation
- All frequently used instructions are executed in 2 to 4 states
- High-speed operating frequency: 5 MHz

Time for +,-,×,÷
- 8±16 bit registers Add/Sub: 0.4 ms
- 8×8 bit multiply: 2.8 ms
- 16÷8 bit divide: 2.8 ms

Low-power operation

Transition to power-down state using SLEEP instruction

Data Structure

Data process
- 1-bit data,
- 4-bit (packed BCD) data,
- 8-bit (byte) data,
- 16-bit (word) data.

Manipulation instructions
- Bit manipulation instructions operate on 1-bit data specified as bit n (n = 0, 1, 2,..., 7) in a byte operand.
- All operational instructions except ADDS and SUBS can operate on byte data.
- The MOV.W, ADD.W, SUB.W, CMP.W, ADDS, SUBS, MULXU (8 bits ´ 8 bits), and DIVXU (16 bits ¸ 8 bits) instructions operate on word data.
- The DAA and DAS instruction perform decimal arithmetic adjustments on byte data in packed BCD form. Each 4-bit of the byte is treated as a decimal digit

Tabella 2.4:*Caratteristichefamiglia Hitachi H8/300 L*

2.2.3 ROM

La CPU ha una ROM da 16kbyte in cui è memorizzato software prodotto dalla LEGO.
L'immagine della ROM è stata quindi ottenuta tramite un processo di riverse ingeneering, la prima immagine completa è datata 1 Ottobre 1998 ad opera di Keoka Proudfoot [Proudfoot 99]
Il compito della ROM è provvedere alle routines di basso livello necessarie per guidare l'RCX ed i suoi sottosistemi grazie ad un driver che si attiva all'accensione del RCX. Il driver è responsabile dell'avvio del firmware[6] (parte memorizzato nella ROM e parte nella RAM), che per default contiene un interprete di by*te code,* il codice con cui viene scaricato ogni programma nel RCX, e con cui comunica il trasmettitore IR.
Le rutine della ROM si occupano di:

- Chiamare la funzione *init_timer* (routine iniziale), che viene inizializzata sul *OCIA handler*[7]
- Controllo dei segnali *on/off/stall* al driver dei motori
- Gestione PWM[8] per i motori
- Gestione dell'A/D per la conversione degli input
- Gestione della comunicazione con porta seriale
- Gestione speaker
- Gestione LCD
- Check dei messaggi di basso livello, come la verifica che l'*opcode*[9] abbia un numero corretto di bytes[10]
- Richiamo del *OCIA handler* ogni millisecondo

La ROM per prima cosa chiama il primo indirizzo del firmware, questo attiva il firmware attraverso un funzione composta da un loop perpetuo. Una volta attivato "userà" la ROM per gestire l'RCX . Le comunicazioni tra ROM e firmware avvengono in vari modi in quanto in essa non sono memorizzate semplici istruzioni ma intere funzioni che possono interagire con tutti i dispositivi dell'RCX
La ROM può essere completamente scavalcata (non letta), se lo si desidera, non chiamando la funzione *init_timer* , inoltre gli *interrupt handlers* presenti nella ROM chiamano indirizzi che risiedono nella RAM quindi anche tutti gli *interrupt handler* possono essere sovrascritti pemettendo un controllo a basso livello del microcontrollore[11]

2.2.4 External RAM

L'RCX dispone di una RAM esterna di 32K indirizzata da 0x8000 a 0xFFFF. Sebbene parte di questi indirizzi siano riservati per la memory mapped degli I/O; in particolare i registri on-chip sono mappati da 0xFF88 al 0xFFFF mentre sono riservati indirizzi per l'LCD display memory, memory mapped motor control, shadow register per le porte I/O ed infine per gli interrupt vectors.
Nella RAM viene memorizzata l'immagine del kernel e i programmi dell'utente ed è quindi divisa in due parti *kernel part* locata nella parte bassa e *user programs part* locata negli indirizzi alti.
Accanto alla memoria esterna l'RCX non presenta nessun altro dispositivo, di conseguenza non vi è la necessità di un filesystem nel sistema operativo.

---

[6] Per ulteriori dettagli sul funzionamento del firmware si rimanda al paragrafo 2.3
[7] funzione di temporizzazione dei dispositivi esterni quali motori, trasmitter, speaker
[8] Pulse Width Modulation: Modulazione ad impulsi, metodo di alimentazione dei motori
[9] Indica il *byte code* dell'RCX
[10] Durante il download del programma nell'RCX
[11] Operazione effettuata dal legOS, si rimanda a Capitolo 5

2.2.5 RCX subsystem

L'Hitachi H8/3292 microcontrollore è equipaggiato con diversi sottosistemi on-chip [Nielson 00]:

- 1 timer a 16 bit: È usato per generare i timer interruptus i quali guidano il Sistema Operativo
- 2 canali con 2 timer da 8 bit: Sono usati per generare i segnali per lo speaker, *timer0,* e per temporizzare comunicazioni del trasmettitore infrarosso, *timer1*
- 1 convertitore A/D: È usato per campionare i segnali analogici provenienti dai sensori
- 7 porte di I/O: Sono usate per monitorare la pressione dei bottoni, il controllo dei motori, gli address bus, le uscite dei sensori,la tensione delle batterie, etc. come si può notare dallo schema riassuntivo in Tabella 2.4 [Proudfoot 99].

NUMERO PORTA	UTILIZZO
1	bit * - address bus LSB
2	bit * - address bus MSB
3	bit * - data bus
4	bit 0 - transmitter range (0=long, 1=short) bit 1 - on/off button input (also irq1 input) bit 2 - run button input (also irq0 input)
5	bit 0 - transmit data (set once, to output, low) bit 1 - receive data (set once, to output, low) bit 2 - external ram power save enable
6	bit 0 - sensor 2 output bit 1 - sensor 1 output bit 2 - sensor 0 output bit 3 - output for sixth handler, set to 1 on init bit 4 - timer output 0, speaker bit 5 - input/output for lcd bit 6 - input/output for lcd bit 7 - timer output 1, infrared carrier
7	bit 0 - analog input 0, sensor 2 bit 1 - analog input 1, sensor 1 bit 2 - analog input 2, sensor 0 bit 3 - analog input 3, battery voltage bit 6 - view button input bit 7 - prgm button input
A/D	A/D is used in scan mode, channel 3 (AN0-AN3), slower conversion time
TIMER 16 bit	free-running timer used for 1 ms clock
TIMER0 **TIMER1**	used for sounds used for infrared carrier
CANALE 1 **CANALE 2**	timer 0 used for sounds timer 1 used for infrared carrier

Tabella 2.5:RCX Subsystem

2.2.6 IR comunication

Le comunicazioni con l'RCX avvengono attraverso la porta infrarossa sia per scaricare programmi o comandare direttamente l'RCX sia eseguire un upload.
La portante è a 38kHz (la stessa dei telecomandi TV)

IR Tower (Trasmettitore IR)

[Proudfoot 99] Per vedere l'interno del trasmettitore infrarosso è sufficiente svitare le 4 viti e rimuovere la cover inferiore (la batteria deve essere preventivamente tolta).

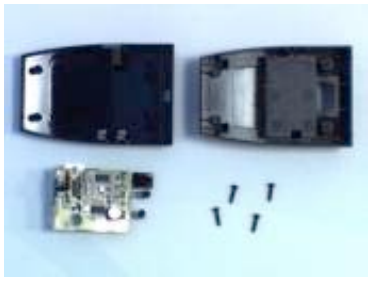


Figura 2.11 Trasmettitore IR smontato

Figura 2.12 intermo del Trasmettitore IR

In Figura 2.11 si nota da sinistra a destra, partendo dall'alto:

- Il coperchio frontale
- Il coperchio inferiore
- Il circuito di bordo
- Le 4 viti

Mentre nella figura più grande vi è il circuito interno. Uno switch ed una resistenza da 6.7ohm sono visibili a sinistra, i 2 chip in mezzo sono banchi di porte NAND ed il circoletto grigio in basso a destra è un regolatore di tensione. Il rettangolo bianco sotto al chip poco sopra il centro del circuito è un LED verde. In alto a destra si nota il ricevitore IR poco più sotto vi sono 2 IR LED identici. A sinistra del circuito , infine, è presente il connettore serale DB-9. Il trasmettitore si autospegne trascorsi 5 secondi di inattività

Cavi di connessione

Il cavo per connetter IR transceiver al PC è un cavo da modem a 6 fili di cui solo 5 sono usati . Numeriamo le hole di ogni connettore da destra a sinistra dell'alto al basso, i pin inseriti dentro queste hole saranno connessi attraverso il cavo ai pin inseriti nelle hole dell'altro capo del cavo nella modo riportato in tabella:

Pin	To	Name	Description
2	3	RD	Receive Data
3	2	TD	Transmit Data

5	5	SG	Signal Ground
7	8	RTS	Ready To Send
8	7	CTS	Clear To Send
4	4		Unusued
1, 6, 9			Have no connection

Tabella 2.6 Configurazione cavo PC-IR trasm.

I segnali RTS/CTS non sono usati per un controllo “grezzo” ma vengono utilizzati dal PC per monitorare se il trasmettitore IR è connesso o meno.
CTS e RTS sono connessi assieme e servono al PC per capire se il dispositivo connesso alla porta seriale sia l’IR trasmitter

Serial protocol
[Proudfoot 98] Il bit encoding è il cuore di un pacchetto del serial protocol. La trasmissione avviene a 2400bps, NRZ, 1 bit di start, 8 di dati, controllo della parità,1 bit di sotp.
Uno ‘0’ è codificato come un impulso infrarosso a 38kHz di durata 417μs , mentre un ‘1’ sono 417μs di niente.
La struttura base di un pacchetto è del tipo:

```
0x55 0xff 0x00 D1 ~D1  D2 ~D2 ...Dn ~Dn C ~C
```

Dove per `D1...Dn` si intende il corpo del messaggio e `C = D1+ D2+...Dn`
In ogni pacchetto trasmesso è presente lo stesso numero di uni e zeri, si può notare come anche la testa del messaggio abbia queta caratteristica ciò permette al ricevitore di “scaldarsi “ prima dell’invio dei veri dati.
La porzione di dati di ogni messaggio inizia con un *opcode* [12] di richiesta se il messaggio avviene dal PC al RCX (0x08 clear), o di risposta , dal RCX al PC (0x08 set), quindi un reply ad una data richiesta è il complemento del codice di richiesta e vice versa. Gli *opcodes* devono mantenere la parità ve ne saranno quindi 2 per specificare ogni richiesta e 2 per specificare ogni risposta, questi due codici differiscono solamente per 0x08 bit
Altre osservazioni sui pacchetti sono presenti in Tabella 2.7

[12] Vedasi paragrafo 2.3 Firmware

```
Basic packet format observations

Immediately after packet header is opcode byte
Remainder is data for opcode
PC sends query opcode
RCX reply opcode is always ~query opcode
RCX completely ignores messages that have invalid packet checksums
Messages sent by PC seem to alternate between having 0x08 set and not set
This 0x08 bit is a sequence bit only in one special case
      - RCX never executes same exact opcode twice in a row
      - Second and beyond are dropped
      - But the same reply that was sent the first time is sent again
      - Toggle 0x08 bit to make sure same opcode twice in a row is accepted
Sometimes packets start with aa ff 00 instead of 55 ff 00
Sometimes packets start with stranger things too
Strange packet headers are most likely due to receiver bias and timing errors
Special packet data 01 02 03 04 ff fe fd fc sent when RCX is not listening
IR Tranceiver  echos commands sent by PC
Sometimes an ff appears a few seconds after a message is received,[Osborn 98] ff
is caused by tower powering down
```

Tabella 2.7 Osservazioni sui pacchetti trasmessi

2.3 Firmware lego

In questo paragrafo non verrà spiegato in maniera intensiva come opera il firmware al basso livello, equivarrebbe a copiare parte del codice presente nella ROM, ma, essendo un interprete di codice è più opportuno mostrare le operazioni che è possibile far compiere all'RCX .
Il firmware una volta attivato si occupa dei 6 *handler* che riguardano:

1. sensori
2. motori
3. tasti e display
4. power e bottone On/Off
5. interprete *opcode*
6. settaggio on/off del bit 3 della porta 6 quando si verifica un init/stop

[Proudfoot 99] Il firmware inizia ad operare all'accensione del RCX ,quando un driver contenuto nella ROM si attiva. Il driver può essere esteso scaricando 16k di firmware nel RCX[13]. Sia il driver che il firmware accettano comandi tramite la porta infrarossa, vi è anche la possibilità di scaricare un programma nel RCX come *byte code* e memorizzarlo in una regione della memoria da 6K, quando accade ciò il firmware interpreta ed esegue il *byte code* del programma .Ovvero [Baum 99] quando viene scritto un programma per l'RCX questo non contiene codice nativo per la CPU, consiste piuttosto in uno speciale *byte code* che viene interpretato dal firmware. Similmente per quanto accade per il *byte code* Java il quale deve essere interpretato dalla Java Virtual Machine.

Opcode e Bytecode

[13] Senza quest'ulteriore firmware no è possibile scaricare programmi all'RCX

Il microcontroller usa un byte code internamente sia per avviare i programmi creati dall'utente e sia come parte di un protocollo seriale per comunicare con il PC
È stata stilata [Proudfoot 99] una lista che descrive l'opcode presente nel byte code dell'RCX, si è visto che lo stesso opcode può avere diversi significati a seconda del contesto in cui è letto, se si tratta di:

- una richiesta dal PC al RCX,
- una risposta dell'RCX al PC
- un comando in RCX byte code

In Tabella 2.10 sono presenti in dettaglio alcune caratteristiche dell'interprete del *byte code*

Il Sistema Operativo lego permette di utilizzare:

- 32k di memoria in totale
- 5 programmi indipendenti nell'RCX, ognuno 6k di lunghezza massima
- 8 Subroutines per ogni programma.
- 10 Tasks per ogni programma
- Illimitate Functions
- 32 storage locations per memorizzare variabili globali (ognuna 16 bit integer)
- Un Datalog per memorizzare dati poi trasferibili al computer ; 3 bytes di memoria per ogni data point (lunghezza illimitata)
- 4 timer con quanti temporali da 1/10th di secondo
- un comando Wait(n) con n espresso in 1/100th di secondo.

Tasks:Ogni task è una sequenza di *bytecode* ,ovvero una lista di istruzioni da seguire. Può essere *attivo* o *inattivo,* nel primo caso l'RCX esegue le istruzioni listate nel task, quando diventa inattivo l'RCX smette di eseguirle.
Quando un programma viene avviato tramite la pressione del tasto "Run" ,il suo primo task è reso attivo e tutti i rimanenti inattivi: fermare il programma (premendo di nuovo "Run"), equivale a rendere tutti i task del programma inattivi.
Possono essere attivati più task allo stesso tempo, in tal caso l'RCX salta da uno all'altro tenendone attivo soltanto uno alla volta per un po' di tempo (time sharing) questo modo di operare è anche conosciuto come *esecuzione concorrente* .

Subrutine: Sono le funzioni all'interno di un Task (max 8 per programma), la differenza rispetto un task è che i task vengono eseguiti in maniera concorrente rispetto ad un altro mentre una subrutine no. Quando un task la chiama, la subrutine viene eseguita ed il task attende finché questa non ha finito prima di eseguire l'istruzione successiva.
Ogni task può chiamare qualsiasi subrutine , ma una subrutine non può chiamare se stessa od un'altra.

Gestione porte output: possono essere attivate in 3 modi *on, off, floating.*
On: i motori sono attivi
Off:i motori vengono bloccati
Floating : i motori non vengono alimentati, e possono ruotare liberamente
Per ogni uscita è prevista una direzione di marcia *forward* e *reverse ,* il senso di rotazione dipende comunque dal verso in cui sono stati collegati i cavi di alimentazione
La potenza fornita può essere settata tra 8 livelli ed avvinene in PWM (modulazione di impulsi), ad ogni livello corrisponde un diverso duty-cicle (percentuale di tempo in cui un motore viene alimentato in un periodo.

Gestione sensori: devono essere prima settati il *sensor type ,* il tipo di sensore che si utilizza (luminosità, bump, rotazione..) ed il *sensor mode* , il modo in cui si utilizza.
Ogni 3 ms vengono letti i valori dai sensori (per dettagli si rimanda la Cap 3 "Sensoristica"), ad ogni sensore sono associati 3 valori

- *Raw value:* È il valore in volt dei sensori quantizzato, varia da 0 a 1023
- *Boolean value:* Può assumere solamente 2 valori 0 ed 1, viene ricavato dal precedente attraverso una tabella ed in base ad un parametro *sensor slope* (con range 0-31) che determina come *raw value* è convertito in bolean

Sensor slope	Condizione	Boolean value
0	raw < 562	0
0	raw > 460	1
0	460 < raw <562	invariato

Tabella 2.8 Boolean (slope=0)

Condizione	Boolean value
raw – raw_old > slope	0
raw – raw_old < -slope	1
raw < (1023 –slope)	0
raw < slope	1

Tabella 2.9 Boolean con parametro slope

In Tabella 2.8 si nota come avviene la conversione in boolean quando lo slope è nullo, la zona di interdizione serve da isteresi in modo da evitare che oscillazioni dell ingresso attorno al valore raw 500 portino a continue oscillazioni del valore rilevato .
Quando lo slope è non nullo, ogni qual volte il sensore viene letto, l'attuale raw value è confrontato con il precedente valore: se il valore assoluto della differenza è meno del slope il valore boolean rimane immutato, se viceversaè maggiore si procede verificando se il raw ha avuto un incremento , allora boolean settato a 1, o se è si decrementato , boolean value settato a 0 , (Tabella 2.9, in cui "cambiamento" indica la differenza).

- *Processed value:* Valore effettivamente usato nel *sensor mode*
  I SENSOR MODE utilizzabili sono 8:
  1. *Raw mode* : ritorna il *raw value*
  2. *Boolean mode :* ritorna il *boolean value*
  3. *Edge count mode* : vengono contate quante volte il valore booleano cambia, cambiamenti più rapiti di 300ms non vengono rilevati (si suppone l'utilizzo in questo "modo" con interruttori)
  4. *Pulse count mode :* simile al precente, il conteggio avviene sui fronti di discesa, quando il valore booleano passa da 1 a 0
  5. *Percentage mode :* convere il *raw value* in n numero da 0 a 100
  6. *Rotation mode* : Utilizzabile per il rotation sensor, restituise un valore da 1 a 16 , per ogni valore si intende un multiplo di 22.5 gradi
  7. *Celsius and Fahrenheit mode:* converte il *raw value* nella temperatura letta

Timer: i 4 timer di cui si dispone misurano il tempo in incrementi da 100ms , possono essere resettati indipendentemente l'uno dall'altro. Il valore massimo a cui possono arrivare è 55 minuti.

Variabili come citato all'inizio del paragrafo è possibile solamente l'utilizzo di 32 variabili globali intere (con valori compresi tra 32767 e –32768). Siccome l'allocazione di memoria per tali variabili è fissa quando si esce dal programma non vengono cancellate , ma potranno essere riutilizzate de un altro programma .
Le operazioni matematiche, con cui modificar le variabili, fornite dal firmware sono addizione,sottrazione,moltiplicazione,divisione, OR e AND (bit a bit), segno,valore assoluto.

Datalog È uno array la cui taglia è definibile dall'utente ,al massimo può essere grande come un programma ,siccome ogni locazione occupa 3 bytes avremo 6k / 3 = 2 circa 2kbytes .Si possono memorizzare valori di variabili o sensori. Per leggere il suo contenuto è necessari eseguire un upload dall'RCX

Power down il sistema Operativo predispone anche lo spegnimento automatico dell'RCX per salvaguardare le batterie dopo 15 minuti. Purtroppo non sono 15 min. di inattività, trascorso questo tempo anche se un programma sta funzionando l'RCX si spegnerà.

In Tabella sono mostrato in dettaglio quanto fin sopra descritto

Byte Code Interpreter

Tasks and subroutines

```
8  subroutines per program
10 tasks per program
```

Memory map

```
The memory map contains addresses, stored as big endian shorts.  The layout
is as follows:
subs : subrutines
prog : program

   index  description

   00-07  prog 0 subs 0-7 start
   08-15  prog 1 subs 0-7 start
   16-23  prog 2 subs 0-7 start
   24-31  prog 3 subs 0-7 start
   32-39  prog 4 subs 0-7 start
   40-49  prog 0 tasks 0-9 start
   50-59  prog 1 tasks 0-9 start
   60-69  prog 2 tasks 0-9 start
   70-79  prog 3 tasks 0-9 start
   80-89  prog 4 tasks 0-9 start
   90     datalog start
   91     datalog next
   92     first free
   93     last valid


The addresses increase monotonically, so the end of the space allocated
for one item can usually be found by looking at the next address in the map.

The first address is 0x0ceba and the last address is 0x0e6b9.  Therefore the
total size of USER MEMORY is exactly 6K.

The memory map says something about how user memory is managed inside the
rcx.  Whenever an item is added, later items are moved up in memory to make
room for the new item.  Whenever an item is deleted, later items are moved
down in memory to consolidate free space.

An empty subroutine takes up one byte of space, while an empty task takes
up zero bytes of space.
```

When a start DOWNLOAD or set DATALOG command is sent, it might fail with an error code related to amount of memory. Memory map was used to confirm this.

Sources

Many commands take parameters as source and argument
Sources are like addressing modes, many of which are unconventional:

```
   0   variable <0..31>
   1   timers <0,1,2,3>
   2   immediate
   3   motor state <0,1,2> <0x07=power 0x08=fwd 0x40=off 0x80=on 0x00=float>
   4   random <return value is in 0 to argument, inclusive>
   5   reserved
   6   reserved
   7   reserved
   8   current program number
   9   sensors <0,1,2>
   10  sensor type <0,1,2>
   11  sensor mode <0,1,2>
   12  raw sensor value <0,1,2>
   13  boolean sensor value <0,1,2>
   14  minutes on clock/watch <0>
   15  message <0>
```

Sources 5, 6, and 7 are never valid.

Loops

The byte code interpreter seems to include a stack of loop counters
The maximum loop counter stack depth is four
Set loop counter pushes stack down and sets the top value
Decrement loop counter and branch decrements counter on top of stack
   Decrement before test
   When top counter is less than zero, stack is popped and branch is taken
   Otherwise branch is not taken
Decrement loop counter and branch only branches forward
   But you can use a second branch to go backwards

Sounds

Sounds seem to be buffered, with buffer size around eight
Sounds are lost when buffer is full
Sounds seem to be asynchronous, meaning their calls seem to return immediately

Programs

Programs use the same opcodes as sent over serial cable
PC opcodes which query state in RCX seem to be treated as noops in programs
All other PC to RCX opcodes seem to work, which is surprising
Unrecognized opcodes cause the program (task?) to halt
Program (task?) also halts when current address is not in valid memory range
   Valid memory range specified in memory map

2.4 Software: linguaggo di programmazione NQC

Il linguaggio fornito nel set base dalle Lego (RCXcode) è di tipo visuale adatto per lo più a bambini e non sfrutta tutte le potenzialità *dell'opcode .*
Si è da subito preferito usare il NQC, Not Quite C , scritto da Dave Baum's [Baum 99] utilizzando il RcxCC 3.1, RCX Command Center [Overmars 99]

Il linguaggio di programmazione è molto simile al C quindi di facile utilizzo, potremmo dire che come il Java è improntato sulle classi il NQC è improntato sui *Task*
TASKS: ogni programma puo'contenere al piú 10 tasks, ognuno ha un nome differente , ma necessariamente uno deve chiamarsi *main* e questo sará il primo ad essere eseguito Gli altri tasks saranno attivi solamente dopo aver usato il comando *start* da questo momento in poi entrambi i tasks saranno simultaneamente attivi. Da un task puó essere fermato un'altro task con il comando *stop*. In seguito questo potra' essere rilanciato con un nuovo *start*
SUBRUTINE: Sono le funzioni all'interno di un Task (max 8 per programma), molto spesso viene utilizzata "come una macro": ovvero tratti ripetitivi di codice vengono accorpati in una subrutine. Lo svantaggio è che vengono memorizzate nella memoria separatamente dal programma e ció potrebbe creare problemi nel caso due task lancino la stessa Subrutine.
FUNCTIONS: sono funzioni che possono o no restituire un valore; la loro versatilitá le rende preferibili in ogni caso alle Subrutine. Ogni programma ne puó contenere un numero illimitato, lo svantaggio (se sono troppo lunghe) è che vengono copiate nel posto in cui sono lanciate e non memorizzate a parte.

Il NQC permette di sfruttare al meglio il firmware della lego, sopportando una programmazione non possibile con l'RCXcode. Dal C prende istruzioni come "define, if, repeat, while, swhtch.." ed introduce innumerevoli costanti per indicare motori, tipi di sensori, accesso al display con un nome dal chiaro significato come "`OUT_A,OUT_B,…,SENSOR_MODE_RAW, SENSOR_MODE_BOOL,…, SENSOR_1,SENSOR_2, …, SOUND_DOUBLE_BEEP, …, DISPLAY_SENSOR_2 …`". Anche il nome dei comandi attinenti all'RCX è molto esplicativo, uno particolarmente importante è il *wait(x)* il quale blocca il task per x millisecondi (mentre i timer lavorano in decimi di secondo), trascorsi i quali riprende dal comando successivo al *wait* . Si ricorda che il blocco del task non provoca lo spegnimento dei motori attivati nel task ,per tale ragione spesso lo si utilizza per far sterzare l'RCX .Oppure per dar modo ai sensori di assestarsi su misurazioni stabili, non oscillanti.
Ulteriori dettagli si possono trovare in Appendice

RCXCC L'RCX Command Center permette la programmazione in NQC in un ambiente di sviluppo dalle molteplici funzionalità . È essenzialmente un editor , dotato di color-sintax code, più file possono essere editati ed è possibile la compilazione ed il download del codice direttamente da questo ambiente, come anche il download del firmware
Le altre funzionalità sono una finestra per vedere all'interno dell'RCX: valore dei sensori, velocità impostata ai motori, valore delle variabili nei 10 task ed eventualmente cambiarli.
Una finestra per controllarlo direttamente con un joystick, scegliendo il motore da attivare, la direzione e la velocità.
Una finestra per la diagnostica ovvero per leggere il livello delle batterie, la porta in cui è connesso l'IR trasmitter , lo stato dell'RCX: vivo se è acceso, morto se è spento .
Una finestra con un piccolo pianoforte con cui suonare lo speaker dell'RCX .
Una finestra per leggere le 32 variabili durante il funzionamento di un programma attraverso un upload temporizzato

Infine una finestra per leggere il Datalog dopo averlo scaricato dall'RCX , durante tale operazione nessun programma deve essere in esecuzione.
Condizione necessaria per tutte queste funzionalità è la presenza dell'RCX acceso, di fronte al trasmettitore infrarosso

Capitolo 3 Sensoristica di base

Sensoristica di base

In questo capitolo vengono descritti i sensori forniti con il set base del ROBOTIC INVENTION SYSTEM e la sensoristica che è possibile ottenete tra l'interazione tra RCX e sensori.
Come già anticipato nella descrizione dei comandi interpretati dal firmware ed in particolare nel comando `sensor_type`, la sensoristica lego riguarda un sensore di luminosità, sensori di contatto, di rotazione di temperatura

3.1 Rilevamento dei sensori

Sia il Touch, il Temperature che il Light lavorano allo stesso modo. La tensione all'ingresso è convertita in un valore quantizzato (`0-1023)` di range `0Volt=0 5Volt=1023`. A seconda poi del tipo di sensore il rilevamento verrà convertito in un numero intelligibile per il programmatore, (Tabella 3.1)
Il Light sensor viene alimentato a `8Volt` (non sono 9Volt , probabilmente dovuta alla caduta di tensione di un diodo), attraverso una resistenza di *pull_up* da `120ohm`, con questa tensione viene alimentato il diodo rosso per circa `3ms;` il rilevamento avviene nei successivi `0.1ms` con modalità analoghe a quelle del Touch o Temperature sensor ovvero attraverso una resistenza di *pull_up* da `10kohm` con alimentazione a `5Volt`.

Volt	Raw	Sensor Ohms	Light%	Temp C.	Touch
0.0	0	0	-	-	1
1.1	225	2816	-	70.0	1
1.6	322	4587	100	57.9	1
2.2	450	7840	82	41.9	1
2.8	565	12309	65	27.5	0
3.8	785	32845	34	0.0	0
4.6	945	119620	11	-20.0	0
5.0	1023	Inf	0	-	0

Tabella 3.1 Valore dei sensori

In tabella si può notare come il Touch sensor assume il *boolean value*[14], il Light viene normalizzato da `0-100` ed è possibile utilizzare le porte di Input non per leggere i sensori ma per rilevare misure di resistenza, alla stregua di un Tester. In particolare se cortocircuitiamo i 2 contatti di un ingresso rileveremo "`0`" mentre se lo lasciamo scollegato (resistenza infinita), rileveremo il valore massimo, ovvero `5Volt`→`1023`

[14] Come descritto nel paragrafo 2.3

3.2 Lego Touch sensor

Detto anche *bump sensor,* il sensore di posizione spesso è costituito da uno switch, (interruttore) meccanico che si chiude quando il sensore viene sollecitato
Nel set Lego ve ne sono 2

Figura 3.1 Touch sensor

In figura si può notare un blocchetto bianco ed uno nero; quello nero è un blocchetto di connessione , la parte simmetrica mancante và collegata alle porte di input; l'unica differenza da un lego classico sono i connettori metallici nella parte superiore , quelli che permettono le trasmissione del segnale elettrico.
Il sensore di contatto è il blocchetto bianco, che nell'edizione italiana è presentato di colore nero (Figura 1.1), si può notare essere lungo 3 "unità"[15] e largo 2; nella parte posteriore presenta un foro a croce utile per collegarlo più saldamente ad altri sensori/componenti attraverso un' asta a croce. La parte attiva dal punto di vista meccanico è la sporgenza di colore giallo, quella che in caso di contatto rientra ed una volta terminato torna fuori, la posizione naturale (assenza di sollecitazioni) è quella fuori, mantenuta attraverso una molla .

3.2.1 Applicazioni

È improbabile usare direttamente il sensore per rilevare un contatto, vista la piccola escursione compiuta dalla parte meccanica; usualmente si preferisce collegarlo ad altri pezzi (aste, "antenne", paraurti,…), liberi di sposarsi ed a volte tenuti fermi da elastici, che in caso di urto premono sul sensore. È possibile anche una realizzazione contraria ovvero usare elastici per mantenere sempre premuto il sensore e fare in modo che al momento del contatto la pressione cali e questo possa tornare allo stato naturale; sarà compito del programmatore individuare questi 2 comportamenti[16].
Gli utilizzi più frequenti avvengono nel rilevare ed evitare ostacoli, la parola "evitare ostacoli "è un po' impropria perché l'ostacolo non viene individuato a distanza ,come nel caso di un sonar, ma è necessario un contatto/urto da parte delle aste/protuberanze collegate al sensore
NOTA: Un errore comune è non accertarsi che l'escursione meccanica del sensore sia completa, in tal caso urti sotto una certa intensità non attiverebbero il bump sensor e conseguentemente non tutti gli ostacoli sarebbero rilevati.

3.3 Light sensor

Il sensore di luminosità è un mattoncino lego da 2×4 unità di colore blu superiormente e grigio inferiormente.Nella parte frontale si possono notare i 2 led, quello rosso (per illumi-

[15] Unità di misura della lego, si riferisce al numero delle "cunette" sul lato a cui si riferisce la misura.
[16] Per un esempio applicativo si rimanda al paragrafo 4.2.3 Obstacle Avoidance

nare) , e quello trasparente il vero sensore di luminosità. Il sensore collegato si può vedere in Figura 1.1

3.3.1 Apertura

AVVERTENZA: Si consiglia di non aprire il proprio sensore. L'operazione non è agevole e rompe permanentemente i 2 blocchetti lego che lo tengono uniti, vi è anche la possibilità di danneggiare irrimediabilmente la struttura interna del sensore.

[Susuraibito 98] [Angeli 98] hanno compiuto un operazione di riverse engineering, disassemblando il Light sensor (Figura 3.2) e cercando di capirne il funzionamento. L'apertura è semplice, sul fondo del sensore vi sono 6 "giunzioni"di plastica, di cui 1 in ogni angolo, basta tagliarli, tenedo fermo il cavo dei fili, e poi sfilare l'involucro grigio.

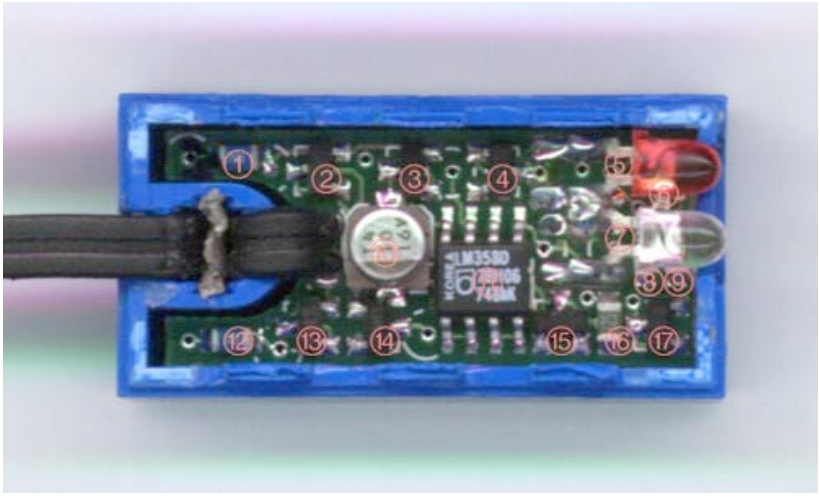


Figura 3.2 Light sensor, aperto

Gran parte dell'analisi si deve a [Gasperi 98] , il quale ha compiuto misurazioni e rilevamenti atti a comprendere meglio il funzionamento di tale sensore.
In figura 3.2 sono evidenziati dei numeri rossi cerchiati, ognuno corrisponde ad un componente od a un gruppo di componenti elettrici, la figura è ancora poco esplicativa , è necessario ricondurci allo schema elettrico equivalente in cui tali componenti vengono riportati e cerchiati di rosso utilizzando la medesima numerazione.

3.3.2 Schema elettrico equivalente: Analisi circuitale

In Figura 3.3 è presente lo schema circuitale del light sensor, i componenti sono cerchiati di rosso,vi sono 3 tipi di numerazione:

- Rossa: la numerazione rossa corrisponde al numero del piedino (pin) dell'integrato che contiene il componente.
- Blu: (numeri tra parentesi) corrisponde ai numeri cerchiati di rosso nella figura del sensore aperto.
- Nera: (numeri decimali) indica il livello di tensione misurato in Volt, in quel punto

D'ora in avanti quando viene indicato un numero si intenderà quelli blu.
Tutti i data sheet dei componenti possono essere trovati nei web site della Panasonic e Philips

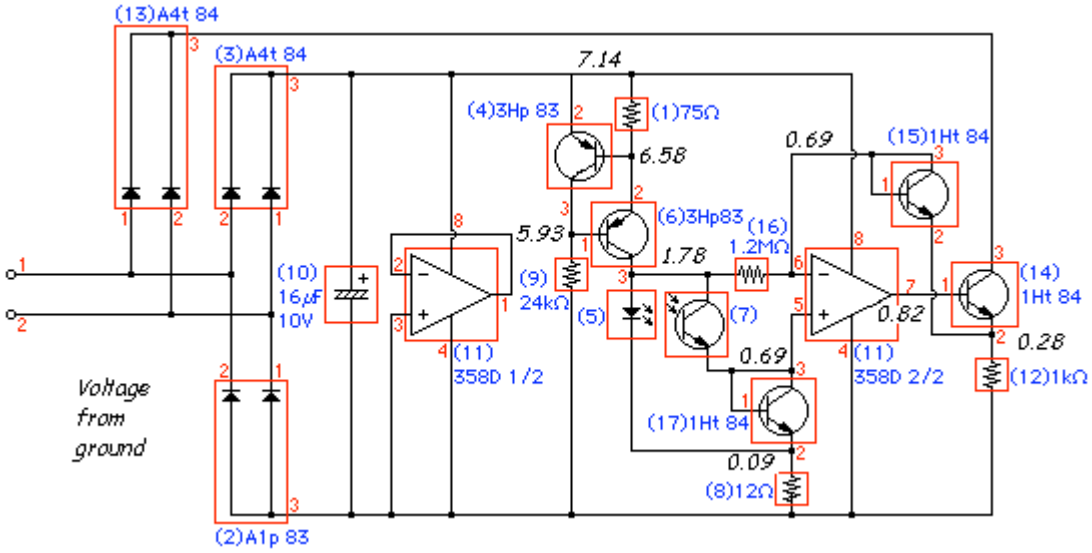


Figura 3.3 Schema elettrico del Light sensor

Analisi circuitale:

`2,3,10` Classico modo di alimentare i sensori utilizzato dalle Lego. `A4t84` è un `BAV70` e A1p83 è un BAW56 doppio diodo ad alta velocità della Philips con una VI[17] curva molto simile ad un `1N4148`.

`11 1/2` È un amplificatore operazionale non usato dell'integrato `LM258`; quest'ultimo è un doppio amplificatore Operazionale, a bassa potenza della Philips a anche National Semoconductor

`1, 4, 6, 9` Servono a tenere a 7.5mA la corrente per il Led 5 e per il fototransistor 7. I due `3HP83` sono `BC857` tarsansistor al silicio, PNP, di utilizzo generale della Philips; il fototransistor probabilmente è un Panasonic `PN168`, al silicio, NPN , il quale ha una sensibilità massima di 800mn. Un interessante effetto secondario del circuito è che se il fototransistor è esposto a una luce molto brillante ed intensa come quella di un laser, il Led si spegne.

`11 2/2` La tensione sul Led è usata come *bias* per il secondo l'amplificatore operazionale `LM358`

`17` Il transistor i questione, usato come diodo, si suppone estenda il range di luminosità del sensore. L' `1Ht84` è un `BC847W`, al silicio, NPN, per uso generale della Philips . Mentre la corrente si incrementata nel fototransistor, la curva Tensione-Corrente del diodo limita la caduta di tensione e estende il limite di luminosità rilevabile.

`15, 14, 12` Anche il transistor `15` è usato come diodo, nella catena di feedback sostituisce la caduta di tensione del transistor `14` . Entrambi i `1Ht84` sono `BC847W` di silicio, NPN, della Philips. Il `14` è un inseguitore di tensione usato per ritornare il valore della luce letto, attraverso il diodo `13` e la resistenza `12`

`13` Come detto sopra permettono di far tornare all'RCX il valore del sensore. L' `A4t84` è un `BAV70` doppio diodo ad alta velocità della Philips

[17] curva VI ovvero curva Tensione-Corrente

3.3.3 Rimozione Led rosso

Se viene rimosso il Led rosso, [Angeli 98] ,il sensore lavora molto bene se usato come sensore di luce ambientale,in Figura 3.4 sono stati plottati gli effetti di tale operazione. Si nota immediatamente come nel caso “con Led” la lettura non arriva mai a zero, questo non perché il Led illumina il fototransistor ma semplicemente perché la sua presenza pregiudica il circuito. Inoltre in assenza di Led è necessaria un po’ più di luce per far partire il sensore e leggere zero. Eseguendo una rilevazione in un a comune stanza si ottiene il 10% ,puntandolo verso una lampada incandescente da 75watt, distante 3 metri d’ombra legge 40% , mentre irradiandolo con un IR Led leggerà 100%
Concludendo i vantaggi sono evidenti, una maggiore risoluzione per la luce ambientale: da 0 a 100 contro una da 20 a 100 di prima; per contro si perde sensitività a luminosità elevata ed l’abilità di poterlo usare come sensore per luce riflessa.

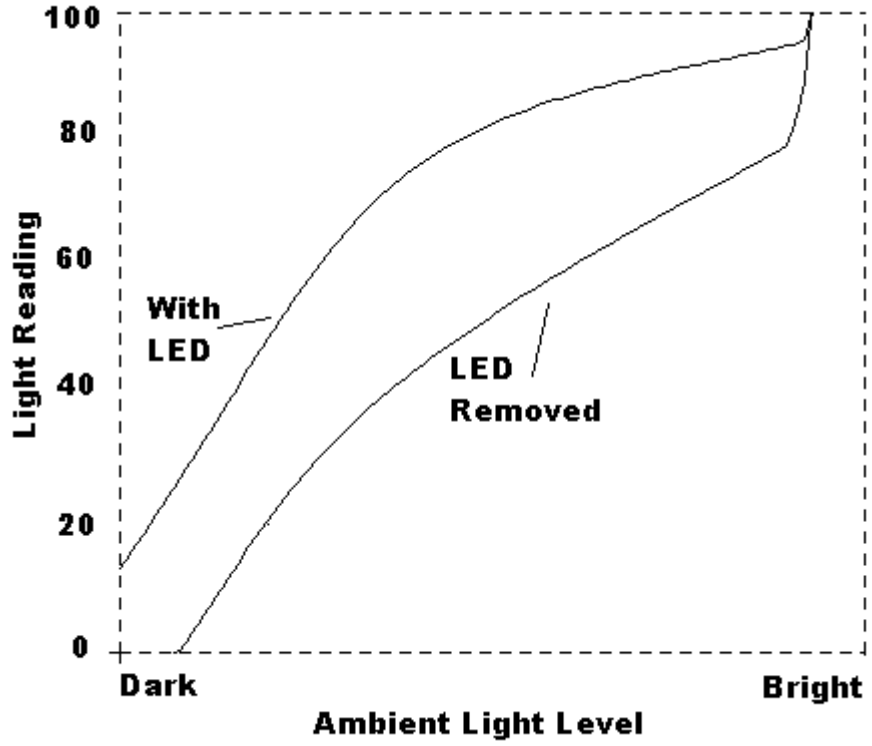


Figura 3.4 Rimozione del led rosso

3.4 Proximity sensor

Il sensore di prossimità non è fornito dalla lego ma può essere creato utilizzando quelli a disposizione. È necessario il sensore di luminosità ed il trasmettitore infrarosso dell’RCX , non è richiesta alcuna modifica hardware, o saldature esterne.

3.4.1 Costruzione

Si sfruttano due fatti:

- L’alta sensibilità del light sensor in particolare la sua capacita di rilevare l’infrarosso
- La capacità di un segnale infrarosso di rimbalzare su una superficie

Si procede allineando sensore di luminosità in linea con il trasmettitore IR, in posizione simmetrica rispetto a questo. Spostarlo più a destra sinistra rende il sensore di prossimi-

tà solo parzialmente operativo, in quanto , a seconda della direzione di incidenza o allontanamento il light sensor rivelerà oppure no l'IR riflesso.
Se l'RCX rimane frontale , il suo utilizzo servirà per evitare ostacoli posti di fronte ed al contrario del tuoch sensor , per rilevarli non sarà necessario urtarli. Per tale ragione è stato chiamato sensore di prossimità : l'ostacolo viene rilevato quando è in prossimità à del nostro robot.
La distanza di rilevamento può essere considerata *" environment ight sensitive"* ovvero dipendente dalla luce ambientale, maggiore oscurità corrisponde ad un rilevamento più tempestivo; nella configurazione di Figura 3.5 si raggiungono i 30cm in una stanza completamente buia
.

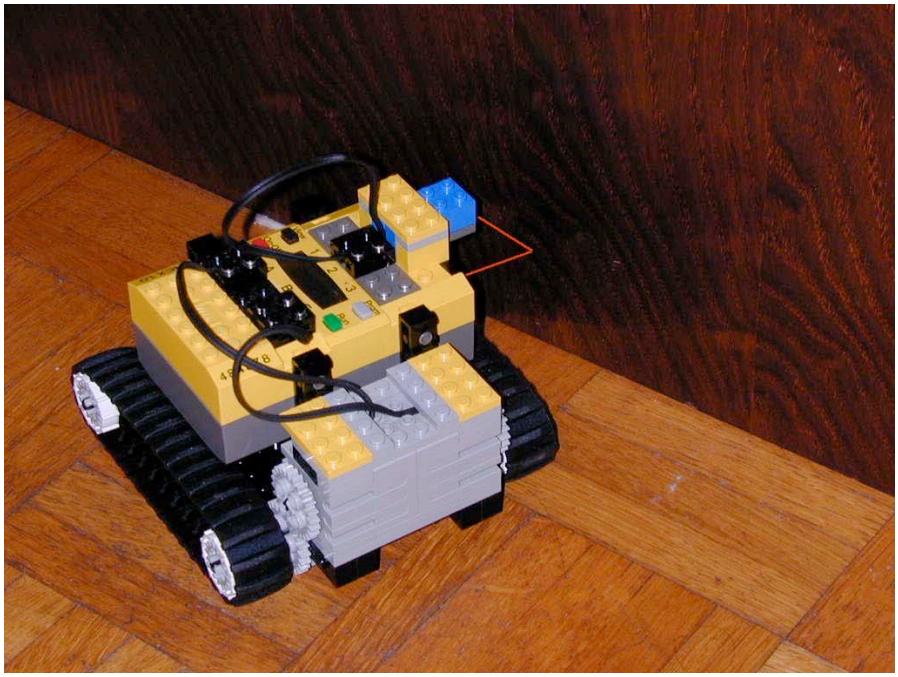


Figura 3.5 Proximity sensor

La configurazione di figura, con RCX ruotato di 90 gradi rispetto al direzione di marcia si è resa necessaria per poter rimanere equidistanti da pareti con le minime correzioni possibili, se si fosse rimasti con sensore frontale , la traiettoria dell'RCX, per fare il check sulla parete avrebbe dovuto essere un susseguirsi di semicerchi.
Per ulteriori dettagli applicativi vedasi Paragrafo 4.2.5 : Follow convex-wall.

3.5 Sensor problem

I sensori sono suscettibili a problemi , alcuni dei quali possono essere compensati via software o hardware oppure dal controllo che vi si applica.
Tipi di errori:

- Glitchy: Rilevazioni spurie
- Noisy: Rumori
- Drifting: Derive di misurazione
- Incorrect range: Sovrastima dl valore degli ingressi da misurare

In Tabella 3.2 è riportato uno schema in cui viene illustrato per ognuno dei 4 problemi, la causa che lo genera, un esempio di robotica in cui accade , le eventuali soluzioni da intraprendere per risolverlo.

	CAUSA	ESEMPIO	SOLUZIONI
GLITCHY	RANGE DEL SENSORE, TEMPO DI RISPOSTA	Il robot si muove troppo veloce ed il sensore o rileva in modo errato o non riesce a rilevare	- Via Software - Diminuzione velocità
NOISY	IPERSENSIBILITÀ STRUMENTALE, CHE RILEVA INGRESSI SPURI	L'elettronica è troppo veloce rispetto alle variazioni meccaniche	Via software, filtrando gli ingressi eliminando i "rumori di fondo"
DRIFTING	CONDIZIONAMENTO AMBIENTALE	Il rilevamento dei colori è modificato dalle variazioni ambientali	Sviluppare capacità di autocalibrarsi, cambiando la soglia dei propri threshold al variare delle condizioni ambientali
INCORRTCT RANGE	SEGNALE TROPPO DEBOLI	Ciò che si intende misurare è vicino o sotto alla risoluzione dello strumento	- Via software - Uso di amplificatori come transistor o amplificatori operazionali

Tabella 3.2 problemi dei sensori

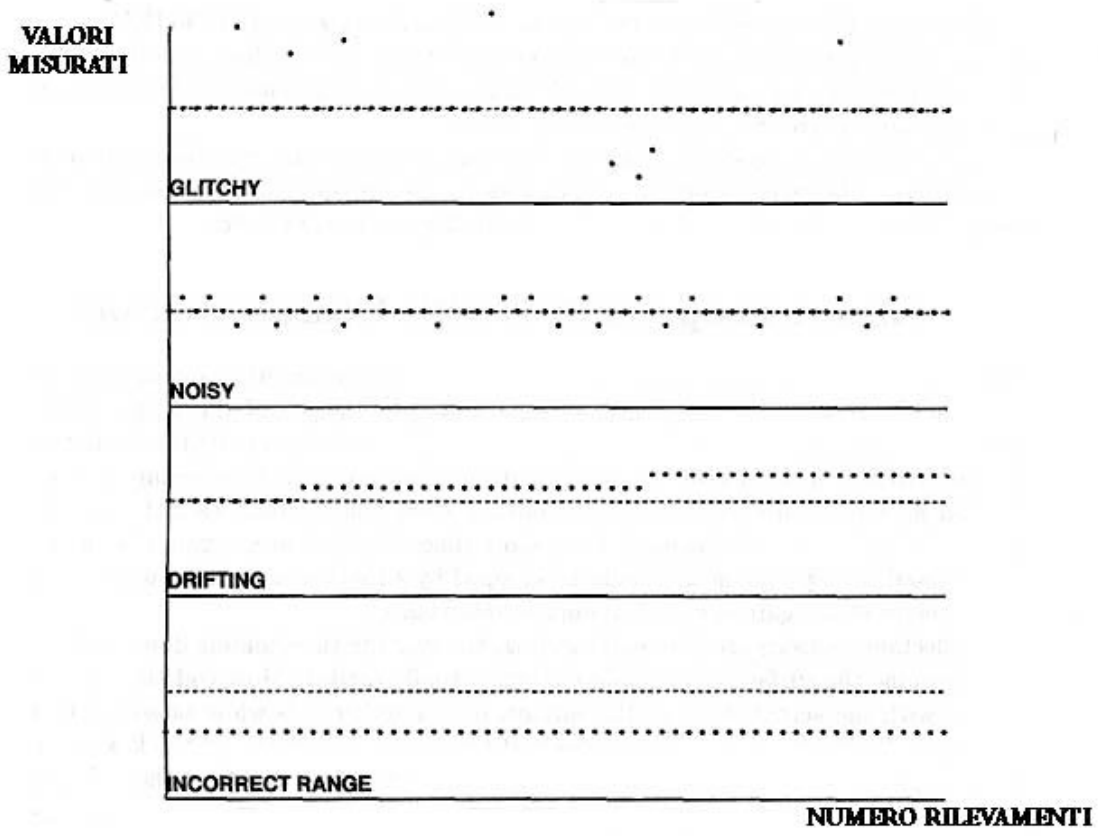


Figura 3.6 Errori

Capitolo 4 Behavior based-control

Behavior based-control

In questo capitolo vengono illustrate da prima le modalità di controllo classico (Feedback) [Mataric 00a],poi le varie architetture di controllo necessarie per gestire più stimoli. Queste partono dal controllo deliberativo (dell'IA-classica) fino ad arrivare al Behavior-based Control passando per il controllo reattivo; non verranno trattati i controlli ibridi, il learning ed il controllo adattativo.
La seconda metà del capitolo è dedicata ad i behavior implementati con la sensoristica fornita dalla lego, infine verranno introdotti altri utili behavior per i quali sarà necessaria altra sensoristica o nuovi pezzi lego.

4.1 Controllo classico

4.1.1 Open loop control

Il modo più ovvio per controllare un robot è fornirgli un task da eseguire, senza alcuna conoscenza dell'ambiente che lo circonda. Questo tipo di controllo viene chiamato *open loop* , a catena aperta. Il robot non compie alcun check, come verificare se l'azione che doveva effettuare è stata completata, i suoi movimenti sono un ordine preciso di azioni. Ogni azione , infatti, consiste in un segnale interpretato dal microprocessore, guidato dall'attuatore e creato dall'output.Figura 4.1

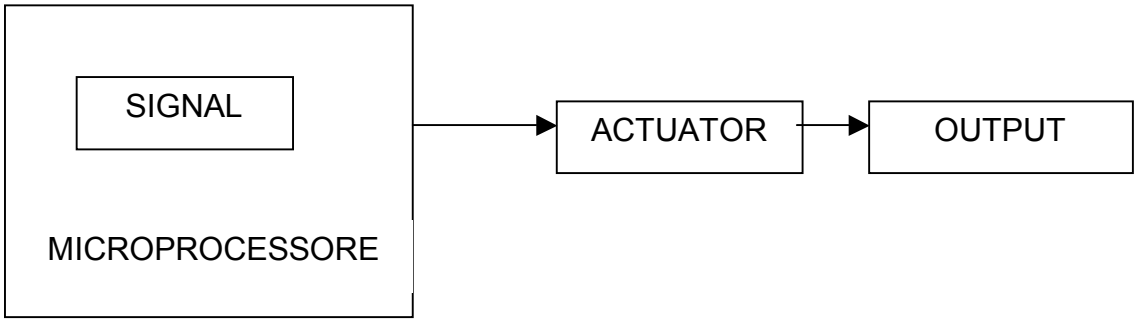


Figura 4.1 Open Loop control

Non c'è collegamento tra l'uscita ed il movimento del robot , il microprocessore non ha idea di dove si trova attualmente. Per farlo spostare da un luogo all'altro bisogna impartirgli comandi del tipo:

```
avanti per 2 secondi                ( per uscire dal box)
gira a destra per 0.8 secondi       ( per sterzare di 90° )
vai diritto per 10 secondi          (prosegue di lato al muro)
aspetta per 3 secondi
…
gira a destra per 0.8 secondi       (sterza di altri 90° )
```

Come si può notare c'è un stretto legame con il tempo e tutti i problemi che potevano verificarsi durante lo spostamento vengono ignorati. Si presuppone implicitamente che ogni

qual volta la procedura viene ripetuta le condizioni ambientali e strutturali rimangano immutate. Infatti non si tiene conto di eventuali asimmetrie dei pneumatici o del livello delle batterie (necessari per far compiere un angolo di 90°), oppure della posizione iniziale del robot o della presenza di altri veicoli durante il tragitto.
Si nota come l'uso dell'*open loop* sia sconveniente. Attualmente viene impiegata solamente in 3 casi:

- Inesperienza di programmazione: costruire un controllore ad anello aperto è semplice
- Limitata sensoristica: mancano i sensori necessari per monitorare gli stati, oppure non sono sufficientemente precisi.
- Controllo *Feed Forward:* è un tipo di controllo in cui conoscendo esattamente il tipo di disturbo e la sua entità, modifichiamo il comportamento del robot affinché questo non abbia effetto, attraverso una catena *open loop*.

4.1.2 Closed loop control (Feedback)

Dato un sistema (nel nostro caso un robot), si cerca raggiungere e mantenere un desiderato stato attraverso continui confronti tra lo stato corrente e quello desiderato.

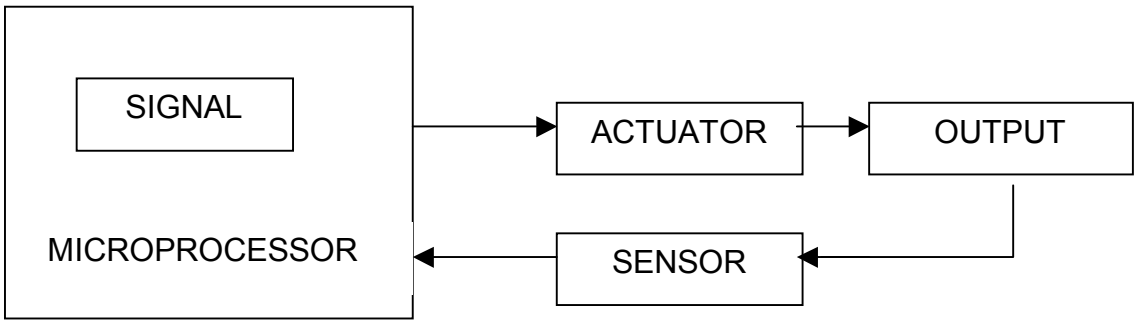


Figura 4.2 Closed Loop control

Lo stato desiderato viene spesso chiamato *goal state* del sistema. Và osservato che questo può essere sia uno stato interno che esterno: nel caso di controllo del livello delle batterie interno, mentre rilevare la distanza da un muro è esterno.
Se lo stato desiderato e quello raggiunto sono gli stessi, il controllo non fa nulla, viceversa decidere cosa fare dipende da come ho progettato il controllore; la differenza tra i due stati viene chiamata *errore*, un altro modo per raggiungere il *goal* è minimizzare l'errore.
Comunemente le uniche informazioni di cui dispongo è l'errore, se è zero si è raggiunto il *goal,* altrimenti in base alla sua grandezza è possibile determinare quanto si è lontani dal goal.
Il comportamento di un sistema dotato di feedback oscillerà intorno allo stato desiderato, similmente i movimenti di un robot oscilleranno intorno allo stato desiderato; ad esempio nel *wall following*, usando semplicemente un sensore di prossimità, quando questo non è attivo significa che ci stiamo allontanando dal muro e quindi il robot sterzerà verso la parete; quando il sensore ha un valore troppo elevato significa che siamo molto vicini al muro e sarà quindi necessario allontanarsi. Lo stesso risultato può essere raggiunto anche con l'utilizzo di sensori più semplici come due *bump sensor* posti su un lato del robot ai suoi estremi; quando quello anteriore non è attivo il robot si sta allontanando dal muro, viceversa, quando quello posteriore non è attivo e l'anteriore si, ci stiamo avvicinando troppo. In

entrambi gli esempi la condizione di corretto *wall following* viene raggiunta attraverso una continua oscillazione tra gli stati di "allontanamento dal muro" e lo stato "troppo vicino al muro".

I tipi di controllo si possono dividere in 3 categorie[18]:

- P (Proporzionale): in cui l'uscita è proporzionale all'ingresso
- PD (Proporzionale - Derivativo): in cui l'uscita è la combinazione di un controllo proporzionale ed un controllo derivativo[19]
- PID (Proporzionale - Integrale - Derivativo): in cui l'uscita è la combinazione di un controllo proporzionale, integrativo e derivativo

4.2 Architetture di controllo

Per Architetture di controllo si intende *software di* controllo del robot, nessun riferimento è fatto ad architetture hardware.
Ci sono diversi modi di costituire un programma, a seconda della struttura e del principio base su cui facciamo riferimento viene usata una appropriata *robot control architecture*.
Il termine *architecture* assume lo stesso significato che ha in *computer architecture;* nella robotica si intende avere a disposizione un set di blocchi costruttivi o di tools di sviluppo da usare per ottenere una fidata programmazione del robot.

4.2.1 Linguaggi di programmazione e architetture

I linguaggi di programmazione sono "Strumenti di realizzazione" ed in generale essi sono *Turing-universal* e come tali, in teoria, qualsiasi linguaggio potrebbe essere usato per implementare qualsiasi architettura. Infatti ogni linguaggio costituito di sequenze, salti condizionati, iterazioni può computare l'intera classe di funzioni computabili (questa proprietà viene chiamata *Turing Universality*), quindi tutti i programmi in uso sono *Turing Universal* o meglio *Turing equivalent* perciò possono essere utilizzati per implementare qualsiasi architettura.
Nella realtà l'uso di alcuni linguaggi viene reso più o meno facilitato a seconda dell'utilizzo che ne vogliamo fare, alcuni si prestano meglio alla creazione di pagine web, altri per il controllo di robot, come il C, oppure specializzati solo per controllare robot come *Behavior_Language* o *Subumption Language* od il *NQC*

4.2.2 Motivi di scelta di un'architettura

[Mataric 00b] Osservando il comportamento del robot non sempre è riconoscibile l'architettura utilizzata[20], questa gioca un ruolo fondamentale nei robot che devono muoversi in ambienti complessi ed eseguire task altrettanto complessi.

La scelta tiene conto di PROPRIETÀ AMBIENTALI:

- Disturbi
- Totale/parzial e stato di osservabilità
- Stati discreti o continui
- Ambiente statico o dinamico

[18] Per una completa trattazione sui tre tipi di controllo vedasi [Oboe 98]
[19] Nel derivativo l'uscita è proporzionale alla derivata dell'ingresso
[20] Allo stesso modo dato il risultato di una funzione non sempre è possibile determinare con quale linguaggio è stata implementata

- Tempi di risposta di sensori/attuatori

Dal MODELLO/RAPPRESENTAZIONE con cui ho deciso di rappresentare il mondo all'interno del robot :

- Spaziale: metrica, topologia (mappe, spazzi navigabili..)
- Simbolico: decodifica astratta da informazioni a stati
- Oggetti: riconoscimento a oggetti del mondo
- Intenzionale: suddivisione in goal, azioni da intraprendere e pianificazioni
- Interna: memorizzazione stimoli-risposte, autolimitazioni..

E delle PROPRIETÀ DEI TASK del robot:

- Rappresentazione dei behavior: come viene rappresentata l' azione
- Granularità dei behavior: time scale di ogni azione
- Interazione e coordinamento dei behavior: come vengono gestiti gli stimoli-risposte
- Basi per specificare i behavior: se si usa un particolare modello biologico

Alti criteri di scelta sono determinati da:

- SUPPORTO AL PARALLELISMO: abilità dell'architettura di eseguire processi/behavior allo stesso tempo.
- MAPPABILITÀ HARDWARE: quanto bene riesco a mappare e quindi a far interagire la mia architettura con i componenti reali del robot , sensori/attuatori, microprocessori, PLA…
- FLESSIBILITÀ RUN-TIME : capacità di poter eseguire correzioni/aggiustamenti o riconfigurazioni[21].
- ROBUSTEZZA: capacita di riuscire a raggiungere il target anche con singoli componenti inutilizzabili od anche capacità di realizzare piccoli di scostamenti dal target in seguito a piccoli errori sul rilevamento dei sensori.
- MODULARITÀ:capacità di poter aggiungere/togliere task senza pregiudicare o dover modificare anche tutti gli altri, una sorta di scorrelazione
- PERFORMANCE: misura della capacità di riuscire a raggiungere l'obbiettivo imposto.

Qualunque sia la scelta, attualmente si ritiene che le *control architectures* si possano dividere in 4 classi:

1. **Deliberative:**
   - Osservazione
   - Pensiero/pianificazione
   - Azione.
2. **Reactive:**
   - Nessuna Osservazione
   - Pura reazione
3. **Behavior-based:**
   - Pensiero distribuito sull'azione
4. **Hybrid**
   - Combinazione di 1+2
   - Pensiero lento
   - Reazione veloce

[21] Importante per il controllo adattativo ed il *learning*

4.2.3 Controllo deliberativo (action-centered)

Usato nelle tradizionali *AI-style robotics* è di tipo *action-centered*. Un input dai sensori viene convertito in un patern che ne rappresenta la descrizione simbolica; l'azione viene pianificata ed infine inviata ad un comando agli attuatori. La pianificazione dell'azione da compiere viene compiuta da un'unica entità in modo centralizzato, da qui il termine *action-centered.*

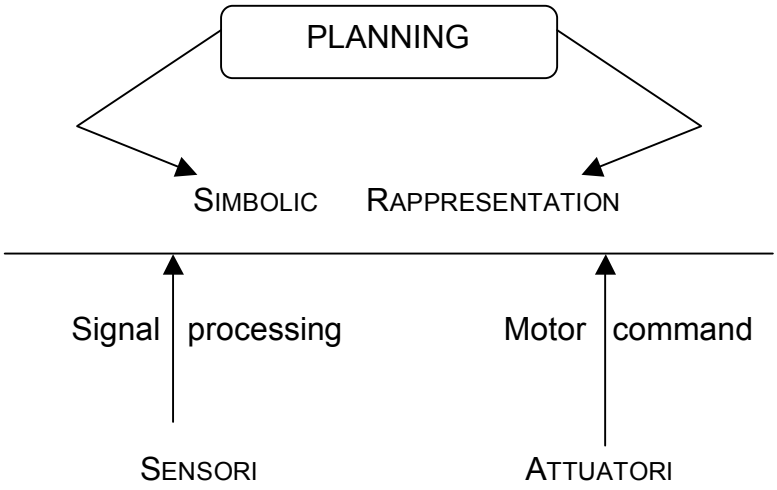


Figura 4.3 Architettura di tipo Action-centered

[Stell 94b] Risulta immediatamente evidente la difficoltà rappresentata dalla costruzione simbolica e dal riconoscimento di questi pattern. Per esempio si consideri l'utilizzo di un IR sensor, il segnale ricevuto varierà dalla distanza dagli ostacoli, dalla luminosità ambientale e dal tipo di superficie del materiale; in ognuno di questi casi il segnale non contiene tutte le informazioni che sono necessarie per un univoco riconoscimento indipendente dal contesto in cui mi muovo.
Altre critiche mosse a questo tipo di controllo riguardano le regole con cui viene decisa la strategia, queste vengono stabilite dal progettista , dopo avere analizzato le possibili situazioni che il robot potrebbe incontrare, ma sia il mondo che i robot sono sistemi dinamici e possono cambiare, mutuare anche accidentalmente, è improbabile riuscire a tenere conto di tutti i possibili cambiamenti per tal motivo è necessario lavorare in un mondo chiuso ed in un ambiente altamente controllato.
Tuttora architetture di tipo *action-centered* sono utilizzate con successo nel controllo classico e nelle reti neurali, il loro coretto funzionamento è reso tale dall'aver omesso rappresentazioni di tipo simboliche.

4.2.4 Controllo reattivo

[Arkin 98]. Con questo tipo di controllo si intende una classe di architetture che operano senza compiere osservazioni ed con un *time scale* molto basso.
I sistemi puramente reattivi non usano una rappresentazione interna dell'ambiente e non compiono osservazioni, reagiscono alle informazioni attuali dei sensori.
Il robot quindi reagisce direttamente al mondo così come esso viene percepito dai sensori, evitando la necessità dell'intervento dell'astratta rappresentazione della conoscenza; l'attività sensoriale infatti è in grado di escludere completamente (o quasi) la rappresentazione della conoscenza per applicare un determinato controllo di basso livello degli attuatori (motori).Per chiarire ulteriormente questo concetto si riporta una frase di [Arkin 98]. "*In*

essence, what you see is what you get". Ciò è particolarmente vero in ambienti che variano dinamicamente dove è impossibile ed inutile cercare di prevederne l'andamento. Il fatto di tentare di modellare il mondo oltre a consumare tempo di computazione, causa errori di previsione compromettendo la potenziale correttezza delle azioni che il robot deve eseguire.
Una caratteristica di questi sistemi è essere inerentemente modulari (se visti da un'ottica di progettazione software),infatti permettono ad un eventuale progettista di espandere le "competenze e le capacità" del suo robot aggiungendo nuovi behavior senza ridisegnare tutta la struttura di controllo; caratteristica rilevante se si lavora su sistemi robotici complessi.
In Figura 4.4 vengono evidenziate in maniera schematica quali siano le differenze fondamentali fra architetture deliberative e quelle reattive .

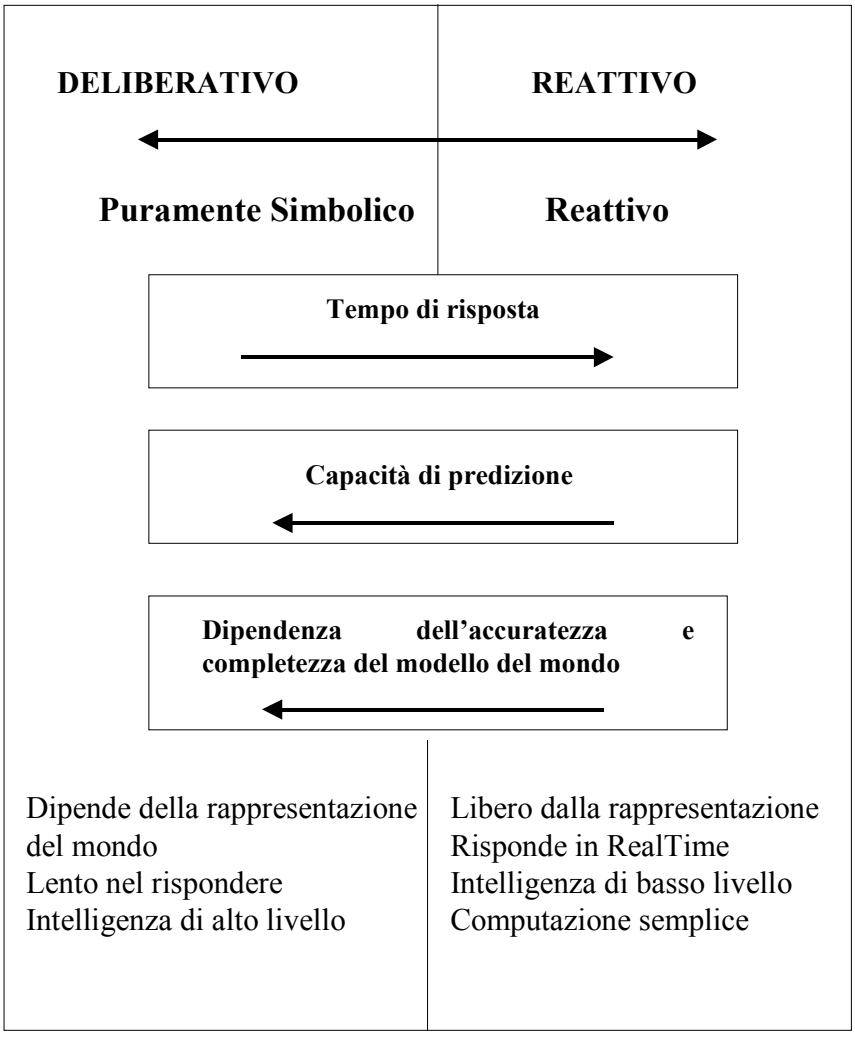


Figura 4.4 Confronto tra Controllo puramente reattivo e deliberativo

Mutua esclusione
Come ribadito sopra la percezione del mondo è mutuamente esclusiva, ovvero viene percepita un unica situazione, nel caso di sensori multipli il controllo dovrà decodificare regole per tutte le possibili combinazioni di ingressi ed ad ognuna associare una singola reazione. Tale mappatura può assumere dimensioni esponenziali è quindi opportuno realizzarla a priori , non in *run-time,* in una *look-up table*; conviene anche utilizzare metodi di ricerca paralleli per ridurre i tempi di ricerca. Comunemente in sistemi realizzati "a mano" non sono considerate tutte le azioni, ma solamente alcune e per tutte le restanti vengono generate azioni di default; ciò semplifica notevolmente la gestione del il sistema di controllo

Arbitraggio
Se il metodo della mutua esclusione non viene intrapreso, più azioni possono essere sviluppate in parallelo, per decidere quale eseguire viene utilizzato un sistema di arbitraggio.
Questo può essere basato su:
- Una gerarchia pre-fissata
- Una gerarchia dinamica
- *Learning* (apprendimento) la priorità viene imparata dinamicamente

Se il sistema necessità parallelismo allora il linguaggio di programmazione dovrà supportare l'abilità di operare in *multi-tasking* , l'assenza di questa potrebbe portare alla perdita di reazione. Infatti se uno stimolo non viene letto, perché il sistema sta leggendo altri sensori, la reazione non verrà generata ed il sistema non può più considerarsi reattivo.

4.2.5 Subsumption architetture
È l'architettura di tipo reattivo più conosciuta introdotta da [Brooks 85], i principi guida sono:
- Il sistema è costruito in modo buttom-up
- I componenti non sono moduli funzionali
- I componenti possono essere eseguiti in parallelo
- I componenti sono organizzati in strati ,dal più basso (*buttom*) al più alto *(up)*
- Gli strati più bassi gestiscono i task più elementari
- Possono essere aggiunti nuovi componenti, che utilizzano gli strati già esisteni
- Ogni componente genera un accoppiamento stimolo-azione
- Non vi è la necessità di un modello interno

Quando si indica che uno strato superiore può utilizzare quelli inferiori si intende la capacità di inibire l'uscita o sopprimere gli ingressi degli strati inferiori.
Nella versione originale sono stati usati le FSM[22], per essere precisi gli AFSM[23].; un AFSM in un certo momento può essere in uno stato, può ricevere uno o più input e trasmettere uno o più output. Sono tra loro connessi in modo da permettere la trasmissione di messaggi, ingressi ed uscite . La Subsumption è quindi un network di AFSM e questo può essere diviso in strati; per esempio una volta raggiunto un task come muoversi evitando ostacoli questo può essere indicato come strato zero.
Il principio base del Subsumption sta nella frase di [Brooks 91]

" *The world is own best model"*

[22] Macchine a Stati Finiti
[23] FSM aumentati con un numero molto piccolo di stati

Se il mondo può fornire direttamente le informazioni (attraverso i sensori), allora questo è il miglior modo per ottenerle e non vi è più bisogno di modelli/ rappresentazioni interne che possono:

- occupare troppo spazio,
- essere lente da consultare,
- dispendiose da creare
- necessitare di continui aggiornamenti

L'arbitraggio decide che uscita verrà inibita e quali ingressi soppressi, il meccanismo è di tipo prioritario: la regola o lo strato con più alta priorità prende il controllo del AFSM.

4.3 Behavior based-system

[Mataric 00b] I *Behavior based-system* usano i *behavior* come sottomoduli del sistema,sfruttando la cosiddetta *behavioral decomposition* , al contrario del *Subsumption* che utilizzava la *task-oriented decomposition* in cui il sistema consisteva in moduli paralleli che concorrentemente cercavano di raggiungere il proprio task (come evitare ostacoli o seguire un muro…).
I *behavior* possono variare da un sistema all'altro, ma tipicamente presentano le seguenti proprietà :

1. I *behavior* sono controlli feedback

2. I *behavior* raggiungono lo specifico task/goal (per es. *avoid-other, go-home)*

3. I *behavio*r sono tipicamente eseguiti in parallelo, in modo concorrente

4. I *behavior* possono memorizzare stati ed essere usati per costruire un modello/rappresentazione del mondo

5. I *behavior* connettono direttamente sensori ed attuatori (prendendo come input il valore di un sensore e mandando l'output direttamente agli attuatori).

6. I *behavior* possono ricevere come ingresso un dato da un altro *behavior* e spedire l'output ad un altro *behavior* ancora (questo segue dalla costruzione del network)

7. I *behavior* sono tipicamente azioni ad alto livello (*go-home* invece di "gira a destra di 37,5° ")

8. I *behavior* sono tipicamente cicli chiusi (*loop*), estesi nel tempo (loop infiniti)

9. Quando sono assemblati in rappresentazioni distribuite, i *behavior* possono essere usati per compiere un'osservazione del mondo ma solamente per un *time-scale* compatibilie con il resto del *behavior-based system.*

Come si è già ricordato questa architettura, non ha i limiti dei sistemi puramente reattivi e presenta le seguenti proprietà :

- L'abilità di reagire in real-time
- L'abilità di usare un rappresentazione per generare efficienti *behavior* (non solo reattivi)
- L'abilità di utilizzare una struttura uniforme per la rappresentazione di tutto il sistema (in modo da non presentare strati intermedi)

La chiave di Volta risiede nella rappresentazione del mondo (qualsiasi tipo di rappresentazione è ammessa) che può essere effettivamente DISTRIBUITA sulla struttura a *behavior.* Per evitare le insidie dei sistemi deliberativi, tale rappresentazione deve agire su un *time-scale* limitato, al più lo stessa parte di *real-time* che ogni behavior ha a disposizione ,in cui è stato diviso il sistema. Invece , per evitare quelle dei sistemi ibridi, la rappresentazione necessita della stessa struttura sottostante dei *behavior* , identica a quella del resto del sistema
Va osservato che non è necessario la presenza di una rappresentazione in ogni punto dell'architettura, possono esistere anche componenti reattive, l'importante (per rimanere *behavior-based system)* è l'utilizzo di *behavior* e non di regole.

Behavior
Oltre ad essere le basi dei *behavior-based system,* possono essere utilizzati per indicare anche qualcosa di osservabile ovvero il comportamento del sistema/robot. Esistono quindi *behavior* interni ed esterni; non necessariamente coincidono, per esempio nella robotica reattiva i comportamenti esterni vengono generati da una collezione (strati) di regole interne. Nella *behavior-based* questo accade, almeno per i *behavior* che si occupano di far eseguire al robot azioni elementari (quelli a basso livello), per gli altri sarà il cloro coordinamento a far scaturire un comportamento emergente.

Rappresentazione dei behavior
Realizzato il task che produce il *behavior* esterno desiderato, un *behavior* può essere rappresentato come :
- Diagramma Stimolo –Risposta (SR)
- Notazione funzionale
- Macchina/automa/accettare a stati finiti (FSA)

4.3.1 Diagramma stimolo-risposta
È il più intuitivo e meno formale dei metodi di rappresentazione. Ogni *behavior* viene rappresentato da uno stimolo e da una risposta da esso computata (Figura 4.5).

Figura 4.5 Diagramma Stimolo-Risposta

Per meglio capire come avviene questa rappresentazione viene di seguito riportato un esempio di navigazione in classe.
NAVIGAZIONE IN CLASSE: [Arkin 98] Consideriamo uno studente che debba andare da un'aula ad un'altra. Il compito sembra semplice (almeno per un uomo), ma esaminiamolo più in dettaglio, suddividendolo in task più piccoli; questi sono:

1. Raggiungere la destinazione a partire dalla posizione attuale
2. Non scontrarsi con alcun ostacolo che si trovi lungo il percorso
3. Capirsi velocemente con altri studenti che hanno intenzioni simili o differenti
4. Individuare il cammino e mantenersi sempre sulla destra

per eseguire questo task di navigazione, si sono impiegati quattro differenti *behavior;*
il diagramma S-R è mostrato in Figura 4.6, si nota come l'uscita di ciascun *behavior* è incanalata in un meccanismo che li coordina[24] in base agli stimoli ambientali che questi percepiscono e genera un comando per motori/attuatori.

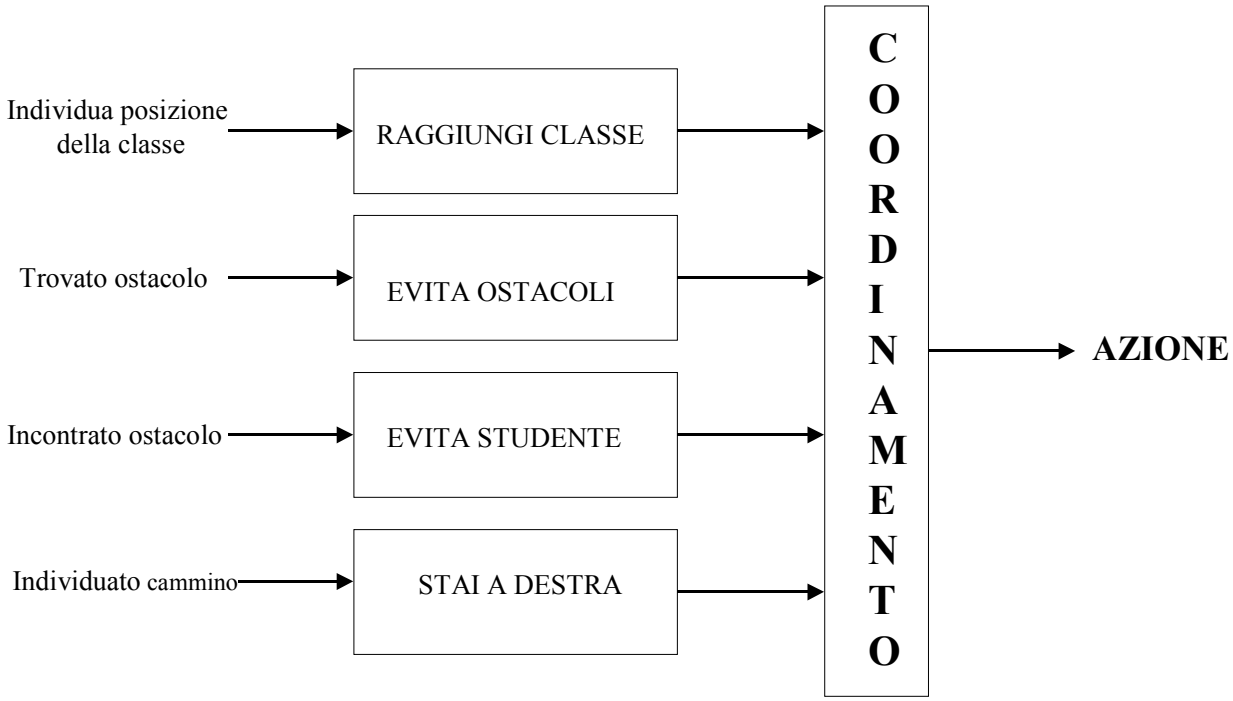


Figura 4.6 Diagramma S-R per problema di Navigazione in classe

4.3.2 Notazione Funzionale

È un metodo tipicamente matematico basato sulla funzione:

$$b(s) = r$$

[24] Per visionare esempi di coordinamento si rimanda a gli ultimi capitoli

In cui significato è: il behavior *b* genera una risposta *r* in presenza di uno stimolo *s*. In un sistema puramente reattivo il tempo non è un argomento di *b* poiché le risposte dei behavior sono istantanee e non dipendono dalla storia del sistema.
Questa notazione è faciel da convertire in linguaggi di programmazione funzionale come il LISP; anche il C è molto popolare ma non è "funzionale" per cui la traslazione non è diretta.
Riprendendo l'esempio Navigazione In Classe (par. 4.2.1), questo apparirà così:

COORDINA – BEHAVIOR [

RAGGIUNGI -DESTINAZIONE (INDIVIDUA- POSIZIONE-DI ARRIVO)

EVITA-OSTACOLI (INDIVIDUA-OSTACOLI)

EVITA-STUDENTI (INDIVIDUA-STUDENTI)

MANTENERSI-A-DESTRA(INDIVIDUA-CAMMINO)

] = RISPOSTA-AI-MOTORI

Quando il tipo di attuatori è univoco, viene tralasciata l'indicazione "= RISPOSTA-AI-MOTORI" .
Sono presentii soliti 4 *behavior* ,ciascuno genera un uscita dipendente dagli stimoli ricevuti dall'ambiente e la funzione principale "COORDINA – BEHAVIOR" valuta queste uscite combinandole in modo opportuno[25]. La "funzione coordinamento" può essere l' argomento di un qualsiasi gruppo di *behavior* ed anche di altre funzioni di coordinamento:

COORDINA – BEHAVIOR [

COORDINA – BEHAVIOR-1(INSIEME-BEHAVIOR-1)

COORDINA – BEHAVIOR-2 (INSIEME-BHEAVIOR-2)
…
]

Dove ciascun insieme di comportamenti è dato dall'unione dei singoli *behavior*. Raggruppando semplici funzioni di questo tipo, in maniera modulare, è possibile costruire architetture molto sofisticate.

4.3.3 Acettore a Stati Finiti (FSA)

Stati e stati di transizione sono molto facili da decodificare e disegnare su un diagramma a stati finiti, gli stati del diagramma possono essere i *behavior* ,in tal modo il diagramma mostra la sequenza delle transizioni dei *behavior*. Tutto questo viene chiamato Accettare a stati finiti [Arbib, Kfoury, Moll 81], ha delle proprietà utilissime per descrivere quali *behavior* siano attivi in un determinato momento , mentre è meno utile per rappresentarne uno singolarmente poiché si riduce ad un FSA banale.
Un Accettare M è una quadrupla così formata:

[25] In generale le uscite dei behavior non sono necessariamente tali da poter essere combinate con quelle di altri in maniera ottimale

$$M = (Q, \delta, q_0, F)$$

dove:

- Q rappresenta l'insieme degli stati possibili per i *behavior*
- δ rappresenta a funzione di transizione che mappa stati e iput in nuovi stati (può essere rappresentata come tabella
- q_0 rappresenta la configurazione iniziale dei *behavior* $q_0 \subseteq Q$
- *F* rappresenta l'insieme degli stati accettabili, un sottoinsieme di Q, che può inviare un comando ai motori

Gli FSA vengono impiegati per specificare sistemi molto complessi , dove insiemi di *behavior* primitivi vengono attivati o fermati durante l'esecuzione di compiti complessi di alto livello.
Con questo tipo di rappresentazione, l'esempio di Navigazione in classe diventa:

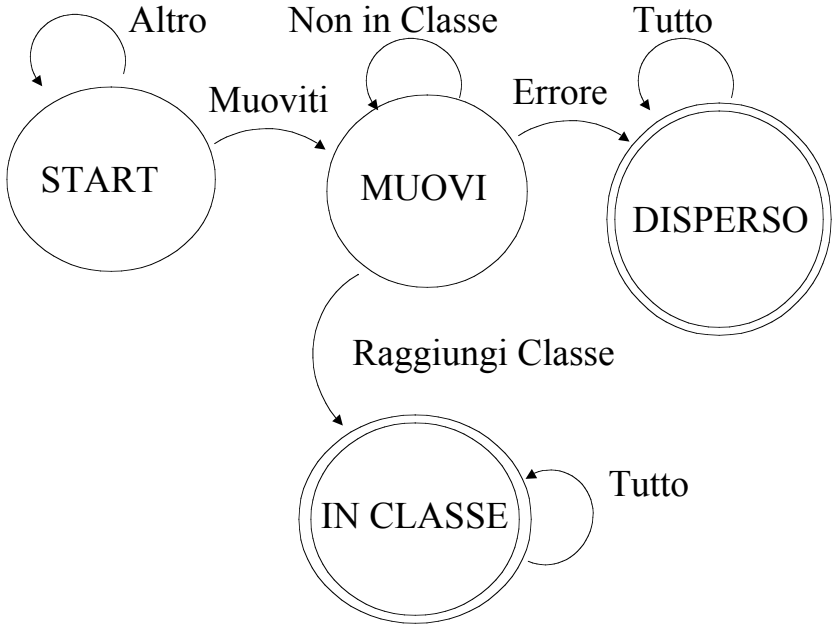


Figura 4.7 FSA per problema di Navigazione In Classe

In figura si possono notare i 4stati: START, MUOVI, DISPERSO, IN CLASSE, gli ultimi due sono stati terminali e graficamente sono indicati con un doppio cerchio. Lo stato DISPERSO non dovrebbe mai verificarsi, se il robot funziona correttamente e se non ci sono imprevisti; mentre lo stato goal è , IN CLASSE (questo dovrebbe essere sempre raggiunto).
Lo stato che racchiude in sé un vero e proprio behavior è, MUOVI, esso comprende infatti 4 *behavior* di livello più basso: RAGGIUNGI-DESTINAZIONE, EVITA-OSTACOLI, EVITA-STUDENTI E MANTIENI-LA-DESTRA e , MUOVI è un *behavior* di coordinamento.
L'acettore sarà quindi del tipo:

M=({*START, MUOVI, DISPERSO, IN CLASSE*}, δ , *START,* {*DISPERSO, IN CLASSE* })

Concludendo possiamo affermare che l'FSA permette un buon grado di astrazione, grazie al quale si riescono ad esprimere le relazioni fra i differenti *behavior* e fra gli insiemi di *be-*

havior. Può essere utilizzato non solo come metodo di rappresentazione ma anche come vero e propria architettura di coordinamento dei behavior .

4.3.4 Encoding

Oltre alla rappresentazione vi è anche il problema di come codificare le uscite dei *behavior* , infatti il loro responso nello spazio fisico ha una forza ed una orientazione, se espressi nel modo Stimolo Risposta possono essere mappati in 2 categorie:

- **Discrete encoding**
  - Espresso come un set finito di coppie Stimolo –Risposta
  - Il mappaggio spesso è associato alla regola base IF THEN
  - Esempi includono: Gapps [Kaelbling & Rosenschein 91], Subsumption language [Brooks],behavior language [Brooks 90]

- **Continous encoding**
  - Viene utilizzata una funzione matematica continua per descrivere il zappamento input-otput
  - Può essere semplice, tempo-variante, armonica
  - Esempi includono: campi potenziali (*vector field,* realizzazione competitiva), schema theory[26] [Arib 92] (*"fusion vector field"*), problemi con minimi/masimi locali, *oscillatory behavior.*

4.4 Behavior Coordination

I metodi di coordinamento dei behavior sono di 2 tipi, Competitivo e Cooperativo, non verranno trattati in questa tesi, ma si desidera comunque effettuare una rapida panoramica

4.4.1 Competitivo

È il coordinamento in cui più comportamenti concorrono in modo competitivo ad utilizzare una risorsa del robot (Per visionare qualche esempio si rimanda al paragrafo 4.5).
Vi è spesso un congegno deputato al cosiddetto "arbitraggio"dei behavior, ovvero è il sistema di arbitraggio che sceglie quale behavior attivare. I sistemi competitivi si differiscono appunto dal divero comportamento del loro sistema di arbitraggio.

- *Priority-based:* Implementazione dell'architettura Subsumption (par. 4.2.5)

- *Fixed:* la gerarchia è fissa, vi è un ordine tra i behavior,quello a priorità più elevata sopprime quelli a priorità più bassa , utilizzando la risorsa comune.

- *Priority-based arbitration:* vi è un organo che arbitra la richiesta da parte dei behavior della risorsa, decidendo di volta in volta a che dare la risorsa

- *Action selection:*viene sfruttata l'inibizione laterale dei Bhehavior, ogni behavior che dovrebbe attivarsi, pesa negativamente i behavior che gli stanno a lato. In pratica il primo behavior che attiva inibisce tutti gli altri, fin quando non ha terminato di lavorare

[26] Basato sulle neuroscienze in particolare sulle scienze cognitive, la rappresentazione è come i campi potenziali, con la differenza che non avviene una competizione ma una fusione. [Arkin 89a],[Arkin 89b]

- *Democratic:* ogni behavior da un numero di Motor response, (comportamenti da intraprendere), l'azione più votata viene eseguita

- *Decision tree:* viene percorso un albero di decisione, la foglia in cui arrivo è il behavior da attivare.

- *Vector fieldI:* campi vettoriali, ogni oggetto nello spazio, è rappresentato da campi vettoriali, ogni behavior viene pesato diversamente, il dominante si attiva

- *Fuzzy*: Viene utilizzata la logica fuzzy per riconoscere strutture geometriche semplici (corridoi, angoli,porte) e con queste conoscenze vengono attivati i comandi ai motori adatti a percorrere quella zona

4.4.2 Coordinamento cooperativo

Si dice cooperativo il coordinamento di più behavior che sono attivi contemporaneamente, non vi è uno che domina l'altro, vi è una fusione delle uscite dei behavior. I primi due sfruttano una combinazione lineare, il Fuzzy invece sfrutta la rappresentazione.

- *Motor Schema:* sfrutta anch'esso i *Vector field,* ma in questo caso le risposte vettoriali dei behavior, per i loro pesi vengono sommate, ed il comportamento seguirà il vettore risultante.È una fusione dei comportamenti. [Arbib 92] basata sulle scenze neurocognitive.

- *Potential field:* Simile al precedente, lo spazio viene rappresentato in campi potenziali, il robot si muove attraverso linee di liveollo

- *Behavior blending (fuzzy):* Attraverso un set di regole viene selezionata un'azione da un set di azioni primitive, la fase di Blending corrisponde alla fuzzificazione, quella di selection alla defuzzificazione.

- *Tropismo:* è una tecnica di Learning, in cui i comportamenti (behavior) cooperano per imparare nuovi behavior

Non si può concludere senza almeno nominare Altri due controlli:

- *Adaptive Behavior0*
- *:* è un coordinamento adattativio in cui i behavior vengono modificati, per meglio raggiungere il goal

4.5 Behavior implementati con sensori di base (Lego sensor)

Con il robot nella configurazione tipo tank, e l'uso del linguaggio NQC sono stati implementati alcuni *behavior,* alcuni elementari, altri più ad alto livello coordinando in modo elementare alcuni a basso livello.
Lo scopo è creare un "parco *behavior*" su cui implementare tutti i metodi di coordinamento dei *behavior* estinti, per implementare task complessi; come vedremo in seguito la limitata sensoristica ed il software limiteranno questa nostra ricerca.
Per ogni *beahvior* verrà indicato:

- Sensori necessari per implementarlo
- Funzionamento
- Architettura di coordinamento impiegata (se presente)
- Diagramma SR (se necessario)
- Codice NQC

NOTA:Il termine *behavior* e task, in questo contesto hanno lo stesso significato di comportamento emergente del robot, da non confondere con la parola task della programmazione NQC, un *behavior* infati puòessere composto da più task.

4.5.1 Move-haead

Sensore Necessario:Nessuno
È il primo dei *behavior* elementari necessari per far muovere il robot, la presenza dei 2 motori è scontata ed indispensabile, il robot è a forma di tank (come per tutti gli altri *behavior*).
In tank compie un movimento rettilineo, generato dall'azionamento dei motori in configurazione `fwd` o `rev` a seconda che si desideri andare avanti od indietro, predisponendo una velocità di crociera eguale per le due ruote.
Questo è il task a più basso livello, quello che in molte applicazioni ha priorità minore e quindi viene interrotto dagli altri più complessi; generalmente è anche quello sempre attivo in un ottica in cui il robot va sempre diritto, poi se incontra ostacoli o riceve determinati stimoli reagisce di conseguenza (in base agli altri behavior), quando questi terminano continua la sua marcia.
Nei sistemi a minima sensoristica si può immaginare di utilizzarlo come una versione semplificata del task *move to goal* in cui il robot è al punto di partenza (A) e deve raggiungere quello d'arrivo (B) se questi sono allineati (indipendente mente da cosa sia presente nel mezzo) si può supporre che il robot abbia pianificato di raggiungere il goal spostandosi su una linea retta.

4.5.2 Turn-always

Sensore Necessario: nessuno
È semplicemente un task che permette al tank di compiere sempre curve verso destra a brevi intervalli; un cigolato per sterzare utilizza 2 maniere: se deve girare a destra o blocca la ruota destra , ma questo rallenta la curva ,facendo consumare molte batterie, oppure fa ruotare la ruota in senso contrario.In quest'ultima maniera il tank gira su se stesso, basterà usare un timer e farlo fermare una volta raggiunta la corretta angolazione[27], e da li proseguire.

[27] Si rimanda al Cap.2 , per i problemi riguardanti la corretta rotazione utilizzando silo timer senza dispositivi di autolocalizzazione

Per rallentare la velocità di rotazione basta spegnere i motori per piccoli intervalli di tempo (decimi di secondo) in modo da rendere gli arresti impercettibili all'occhio umano.
Se invece si desidera, come è il nostro caso, far compiere al tank una curva ,sarà necessario ripristinare il moto rettilineo del tank per alcuni istanti; ovvero per alcuni decimi di secondo far ruote entrambe le ruote nella stessa direzione per poi riprendere a farle girare in versi opposti. Il metodo di spegnere i motori per piccoli intervalli produrrà ancora il rallentamento della curva, mentre l'allungare il tempo di moto rettilineo aumenterà il raggio di curvatura.
Questo behavior ha poco senso preso singolarmente se non quello di far compiere al tank curve con un moto simile a quello di un automobile.

4.5.3 Follow-line

Sensore Necessario: light sensor.
Lo scopo di questo behavior è quello di far seguire al tank una linea curva (nera); per far ciò si utilizza il sensore di luminosità,con il led rosso acceso per illuminare la superficie su cui è presente la linea. Sono stati scelti i colori bianco e nero perché il sensore quando li identifica ritorna il valore massimo e minimo rispettivamente, più alta è l'escursione tra i due e più facilmente avviene il riconoscimento.
In realtà il robot non segue la parte interna della curva (zona nera) ma la differenza di luminosità presente tra la linea nera e lo sfondo bianco stando a cavallo tra le due (Figura 4.8).

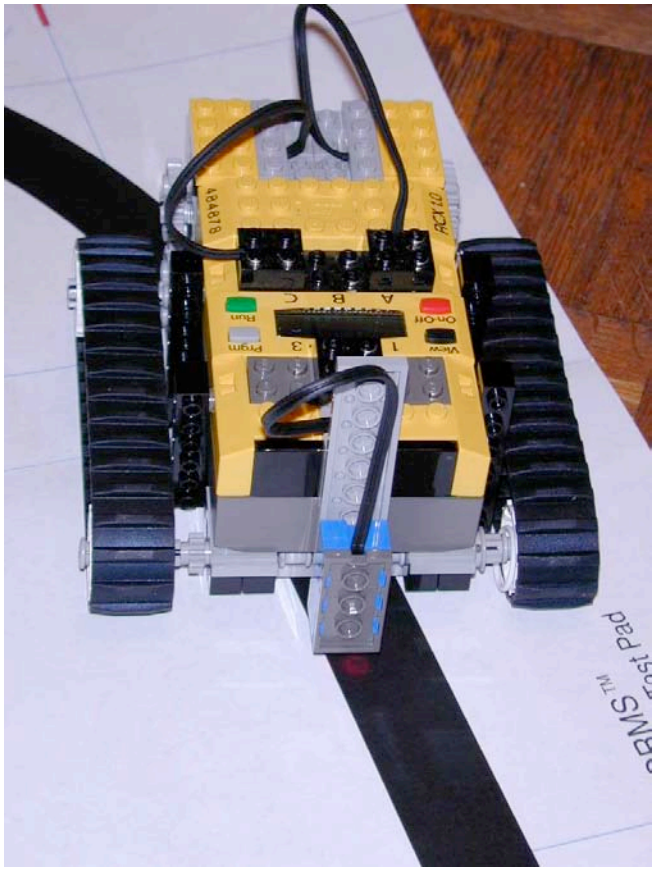


Figura 4.8 Follow-line

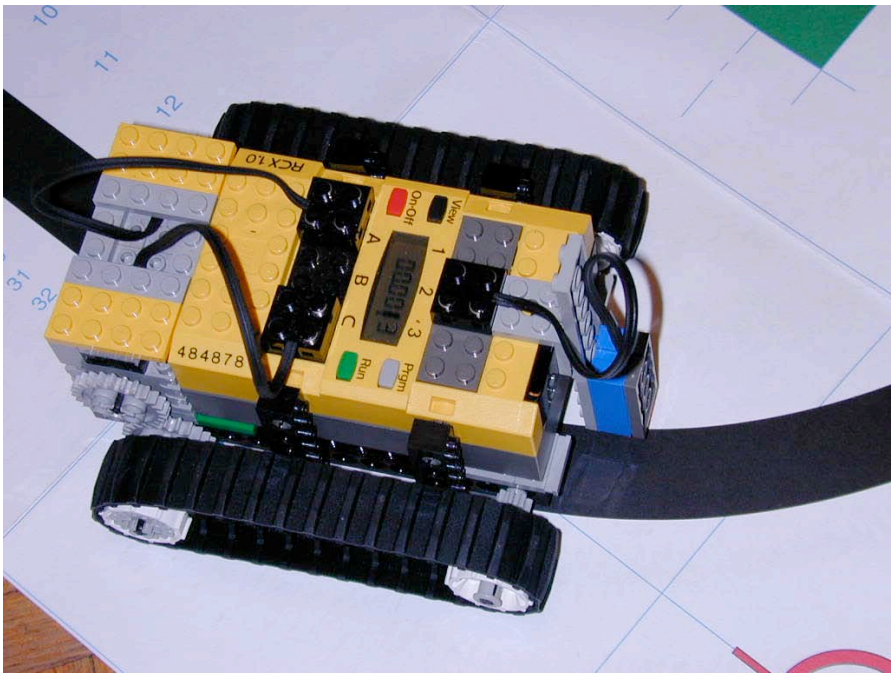


Figura 4.9 Follow-line

Come si può notare dalle figure il sensore di luminosità, usato come riconoscitore di superfici deve sfiorare il pavimento (per non essere influenzato dalla luce ambientale) ed illuminare la superficie su cui opera (usando il led rosso). Nella prima figura ,il tank si sta movendo per uscire dalla linea, questo si nota ancora di più nella seconda in cui il sensore di luminosità è nella zona bianca, il paso successivo sarà sterzare verso destra per tornare all'interno della zona nera.
Il controllo non è a soglia ma a trigger in modo che il robot non vada continuamente a destra e sinistra. Infatti il controllo può essere:

- A SOGLIA : Il diagramma Stimolo-Risposta è un gradino, in asse x c'è il valore del sensore , mentre in ascissa il comando dei motori, il cambio di comando si attiva solamente al superamento del valore impostato (un valore medio tra il nero ed il bianco)

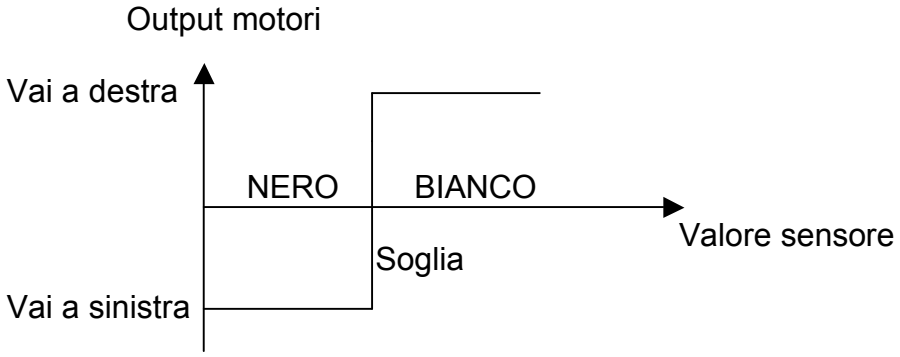


Figura 4.10 Controllo a soglia

L'inconveniente di questo metodo consiste in un moto troppo "scattoso" da parte del tank con continui assestamenti a destra ed a sinistra, questo si evidenzia maggiormente nei tratti lineari.

- A TRIGGER: (od a istreresi). il controllo a trigger risolve questo problema precedente, invece di una singola soglia se ne sono utilizzate 2 , nella “zona morta”il tank continua a spostarsi nella stessa direzione.

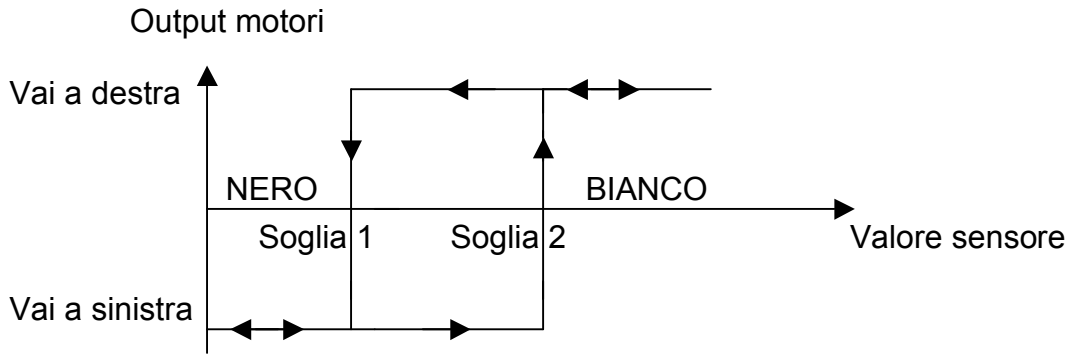


Figura 4.11 Controllo a trigger

In questo modo il tank oscilla molto meno e, in zone lineari può percorrere ampi tratti prima di sterzare. La direzione di percorrenza è univoca: la parte nera sta a destra del tank ,ciò implica direzione antioraria se siamo all’interno della linea, oraria altrimenti. Per affinare questo task, in modo che il tank stia dentro la linea nera è necessario un controllo di posizione che mi indichi da dove stò venendo ed un dispositivo di memorizzazione capace , ogni volta che incontro il bianco, di invertire il moto fino ad allora compiuto;in questo caso oscillerei tra una zona bianca e l’altra, ai lati della linea.

NQC Program

Di seguito viene riportato il sorgente , può contenere commenti in italiano od in inglese. Per chiarimenti sui comandi si rimanda agli appendici.

La prima parte del programma viene utilizzata per inizializzare le due soglie `RIGHT_THRESHOLD` (soglia 2), e `LEFT_THRESHOLD` (soglia 1), facendo correre il tank nelle due zone , prima bianca, poi nera. Passati 2 secondi il tank inizia automaticamente a seguire la linea che stà percorrendo. Le soglie sono date dalla media dei valori rilevati, (quindi vaore massimo e minimo rilevabile), sommate o sottratte di un valore costante.

```
// sensor and motors
#define EYE       SENSOR_2
#define LEFT      OUT_A
#define RIGHT     OUT_C
#define ON(s,v) SetPower(s,v);OnFwd(s);
#define lr LEFT+RIGHT

#define TIME_INIT 20
#define STRAIGHT_TIME 10
int LEFT_THRESHOLD =0,RIGHT_THRESHOLD=0;
/* Si fa muovere il tank per 2 sec. prima sulla superfice chiara
poi su quella scura in modo da auto inizializzare le Soglie
del trigger*/
void init(){
  int i=0;
  ClearTimer(0);
  while(Timer(0)<20){
  RIGHT_THRESHOLD+=EYE;
  i+=1;
  }
  RIGHT_THRESHOLD/=i;
```

```
  Off(LEFT+RIGHT);
  Wait(300);
  i=0;
  ClearTimer(0);
  On(RIGHT+LEFT);
  while(Timer(0)<20){
  LEFT_THRESHOLD +=EYE;
  i+=1;
  }
  LEFT_THRESHOLD /=i;
  RIGHT_THRESHOLD-=5;      //soglia destra
  LEFT_THRESHOLD +=5;      //soglia sinistra
  return;
 }


task main()
{
      SetSensor(EYE, SENSOR_LIGHT);
      ON(lr,7);
       init();
       while(true)
      {
            if (EYE <= LEFT_THRESHOLD)
            {
                  Rev(LEFT);
                  until(EYE > LEFT_THRESHOLD);
                  Fwd(LEFT);
                  Wait(STRAIGHT_TIME);
            }
            else if (EYE >= RIGHT_THRESHOLD)
            {
                  Rev(RIGHT);
                  until(EYE < RIGHT_THRESHOLD);
                  Fwd(RIGHT);
                  Wait(STRAIGHT_TIME);
            }
      }
}
```

4.5.4 Obstacle-avoidance & Corner-detector

Sensore Necessario: 2 *bump sensor*

Questo *behavior* è tra i più conosciuti in robotica, permette di evitare ostacoli con l'uso dei sensori di prossimità ; come si puó notare in figura il tank è stato fornito di due aste (viola) collegate ai bump sensor con degli elastici in modo da essere sempre premuti (livello logico 1), questo sarà considerato il nostro punto di riposo. Nel dettaglio (Figura 4.12) si nota come avviene l'identificazione di un ostacolo: quando si urta un oggetto l'asta si allontana dal bump-sensor potrtandolo a livello logico 0

Il collegamento tra sensori e motori è quasi diretto in quanto se urta l'asta di destra il robot torna indietro verso destra finché il bump non torna a livello di riposo, quando ciò accade i 2 motori riprendono il moto precedentemente interrotto nella stesa direzione. Analogamente accade per l'urto a sinistra.

Questo *behavior* viene utilizzato come comportamento di emergenza da parte del robot durante il raggiungimento di un goal: se c'è un ostacolo verrà superato altrimenti se non se presentano questo task non verrà mai attivato.

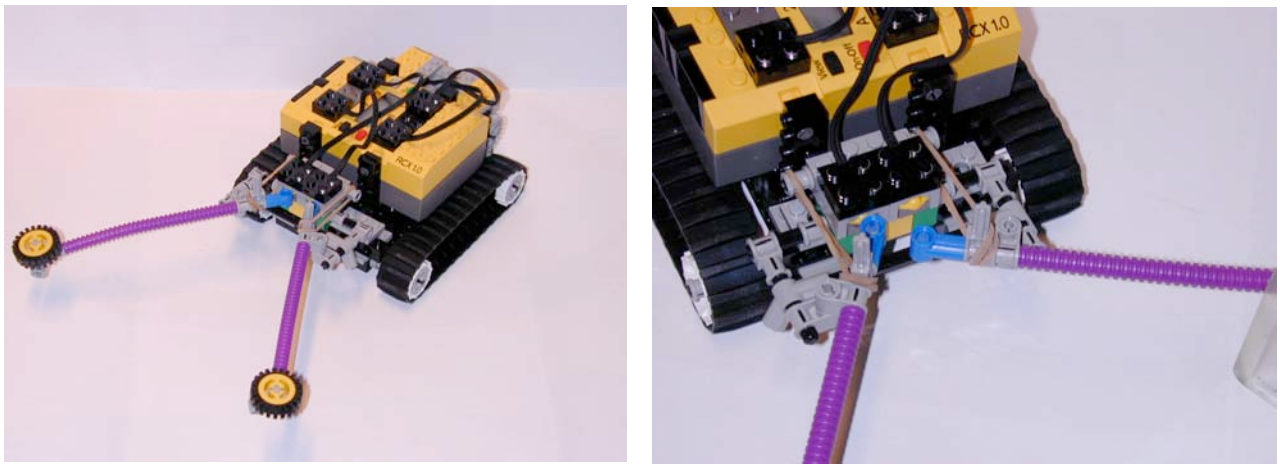


Figura 4.12 Bump sensor in Obstacle avoidance

Corner-detector

L'unico problema delle aste in tale configurazione è la presenza di uno spigolo (un angolo) infatti in questo caso il robot inizierà a spostarsi a destra e sinistra oscillando, "rimbalzando" da un lato all'altro della parete. Per risolverlo è stato introdotto un *Corner-detector ,* ovvero un riconoscitore di angoli.

L'effetto macroscopico è vedere il tank che passato qualche secondo nel rimbalzare "si accorge" di trovarsi di fronte ad un angolo ed inizia a fare marcia indietro allontanandosi per poi spostarsi in direzione di un punto oltre l'angolo.

Questo *Corner-detector* potrebbe essere considerato a tutti gli effetti un *behavior* ,(anche se nel programma viene implementato come una funzione), con un livello di priorità maggiore del semplice *obstacle-avoidance* in quanto interviene bypassandolo e ripristinandolo al suo termine.

La sua libertà d'azione non è così libera come sembrerebbe ma condizionata dell'entrata in funzione del *obstacle-avoidance* ,che può essere considerato il suo *behavior* –padre; quindi ha sì, priorità più alta, ma non può manifestarla finché non è attivo il padre.

NOTA: Per tali ragioni, nel seguito, quando si fa riferimento all '*obstacle-avoidance* si intenderà una versione più fine dell '*obstacle-avoidance* originale essendo fornita del *Corner-detector;* quindi non vi sarà distinzione trai 2 *beahavior.*

NQC program

In questo programma come citato sopra, è stato inserito il *Corner-detector*, per tararlo bisogna settare correttamente le costanti `TIMER_LIMIT` settata a 5 sec e `COUNTER_LIMIT` posta a 10.

Il funzionamento è semplice: se sono avvenute piú di dieci attivazioni dei bump-sensor entro 5 secondi denota la presenza di un angolo; la reazione è allontanarsi e girare verso sinistra.

Il funzionamento normale del task è identico a quello descritto precedentemente; in questa applicazione ruotare indietro, per la configurazione delle aste, viene reso più efficace con il blocco della ruota su cui fa perno la rotazione.

Un'altra particolarità del programma è l'assenza di due task per *check-left()* e *check_right()* (i 2 check sulle aste). È preferibile eseguire un test sequenziale (usando funzioni), dei due bump se non si vuol incorrere nell'arresto del tank quando urtano entrambi i sensori: in tal caso sarebbero stati attivati i comandi `Off(LEFT)` e `Off(RIGHT)` contemporaneamente.

```
// sensors
#define LBUMP SENSOR_1
#define RBUMP SENSOR_3
// motors
#define LEFT OUT_A
#define RIGHT OUT_C

#define TIMER_LIMIT     50
#define COUNTER_LIMIT 10

int counter;

void check_left()
{
      if (LBUMP == 0)
      {
            Off(LEFT);
            Rev(RIGHT);
            until(LBUMP==1);
            On(LEFT);
            Fwd(RIGHT);
            counter += 1;
      }
}

void check_right()
{
      if (RBUMP == 0)
      {
            Off(RIGHT);
            Rev(LEFT);
            until(RBUMP==1);
            On(RIGHT);
            Fwd(LEFT);
            counter += 1;
      }
}

void check_counter()
{
      if (counter > COUNTER_LIMIT)
      {
            // we're in a corner...
            PlaySound(SOUND_DOWN);
            counter = 0;
            Rev(RIGHT+LEFT);

            Wait(100);
            Fwd(RIGHT);
            //Wait(300 + Random(300));
             Wait(23);
            Fwd(LEFT);
      }
}

task watch_timer()
{
      while(true)
      {
            until(Timer(0) > TIMER_LIMIT);
            PlayTone(440, 10);
            ClearTimer(0);
```

```
            counter = 0;
      }
}

task main()
{
      counter = 0;

      // configure the sensor
      SetSensor(LBUMP, SENSOR_TOUCH);
      SetSensor(RBUMP, SENSOR_TOUCH);

      On(LEFT+RIGHT);
      start watch_timer;

      while(true)
      {
            check_left();
            check_right();
            check_counter();
      }
}
```

4.5.5 Avoid-wall

Sensore Necessario: sensore di prossimità
È molto simile all'*obsacle-avoidance* con il vantaggio che no vi è la necessità di un urto con una parte del robot, inoltre l'RCX è stato girato di 90° a destra per questo il *behavior* non ha lo stesso nome.
Si è operata questa scelta per permettere al tank di utilizzare entrambi i sensori, uno per evitare gli ostacoli frontali (*bump)* e l'altro per evitare urti laterali contro una parete,questo evitare di toccare la parete sarà molto utile nei successivi *behavior* .
Il robot opera trasmettendo a determinati intervalli un segnale dalla porta infrarossa, contemporaneamente il sensore di luminosità rileva la luce ambientale , quando a un rilevamento al successivo c'è un grosso aumento di luminosità significa che è stato rilevato il segnale infrarosso, rimbalzato da una parete. Si procede quindi ad una rotazione verso sinistra allontanandosi finché tali rilevamenti non tornano normali.
Per rendersi conto dell'entità dell'incremento basta ricordare che il sensore leggere balori in una scala da 0 a 1023, quando legge un segnale infrarosso la lettura precedente aumenta di 200. (il 20% del fondo scala).

NQC program

.
Si sono utilizzati 2 task uno per generare il segnale IR ,`send_signal` e l'altro per fare il check sul rilevamento del light sensor `check_signal`, quest'ultimo si occupa di far allontanare il tank dal muro finché il light sensor non torna a rilevamenti "normali". È possibile settare l'intervallo di tempo in cui si trasmette (ora a 0.1 secondi) , e il valore di incremento che mi segnalerà la vicinanza di un muro(settato a 200).
Vi è anche un parametro dipendente dalle batterie: la rotazione a sinistra è di 30°, perché viene calcolata facendo ruotare il tank per 0.23 secondi.
Per visionare il codice si rimanda al prossimo paragrafo.

4.5.6 Follow-convex wall

Sensore Necessario: sensore di prossimità.
Questo è il primo *behavior* di rilievo, viene implementato utilizzandone altri 3 finora già visti: *turn-always* e *avoid-wall*
Dal punto di vista macroscopico si vede il tank che segue muri standone ad un distanza massima di 30 cm, ed una minima di 2-3 cm, in stanze particolarmente illuminate. Il movimento è un continuo assestarsi, và verso la parete, si gira se ne allontana (*avoid-wall*), ritorna verso la parete (*turn-always*).
La vicinanza con la parete e conseguentemente il rilevamento è condizionato:

- dalla luminosità ambientale
- dall'angolo con cui mi dirigo verso la parete, più si avvicina ai 90° più tardi rileverò il muro
- dalla velocità del tank , proporzionalità inversa tra velocità ed probabilità di rilevamento

Il comportamento ideale e quello in cui il robot compie pochissime oscillazioni , con un escursione angolare minima: una volta tornato normale il segnale del sensore, il robot inizia a compiere una curva molto ampia, ovvero a sterzare lentamente, come si può notare nel codice
NOTA: Viene usato il termine *convex wall* in quanto il tank è capace di seguire solamente perimetri esterni di figure convesse.
Dal punti di vista apppliatico significa non riesce ad entrare in stanze la cui porta è aperta nella direzione di arrivo del robot

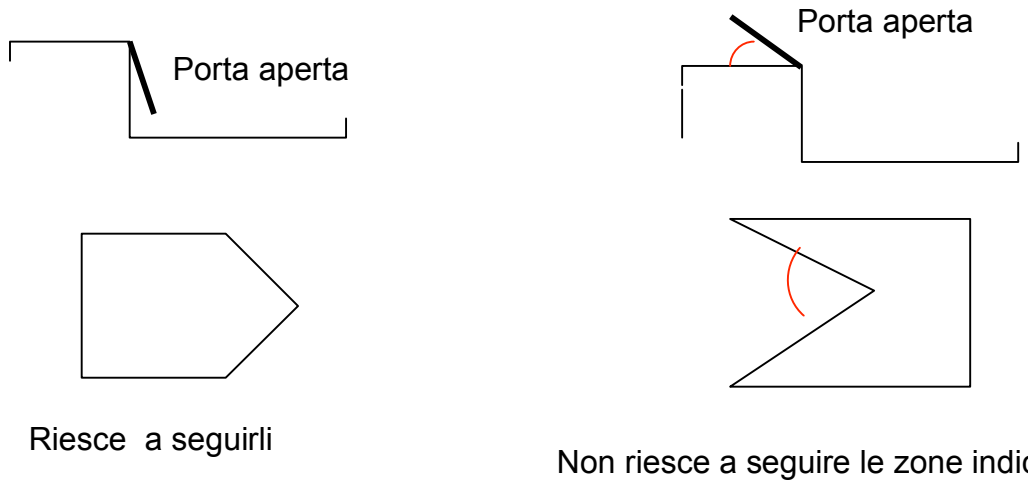


Figura 4.13 figure converre e non

Diagramma SR
Utilizzando più behavior è utile mostrare sul diagramma Stimolo Risposta quale tipo di coordinamento è stato scelto per gestirli.
La prima figura (Figura 4.14) mostra il *behavior* *avoid-wall* diviso per chiarezza nei 2 task indipendenti che lo costituiscono : `send_signal` e `check_signal` ; in cui il primo indica l'emettitore di segnale IR,sempre attivo, ed il secondo quello che lo rileva il segnale e si occupa di allontanarsi.

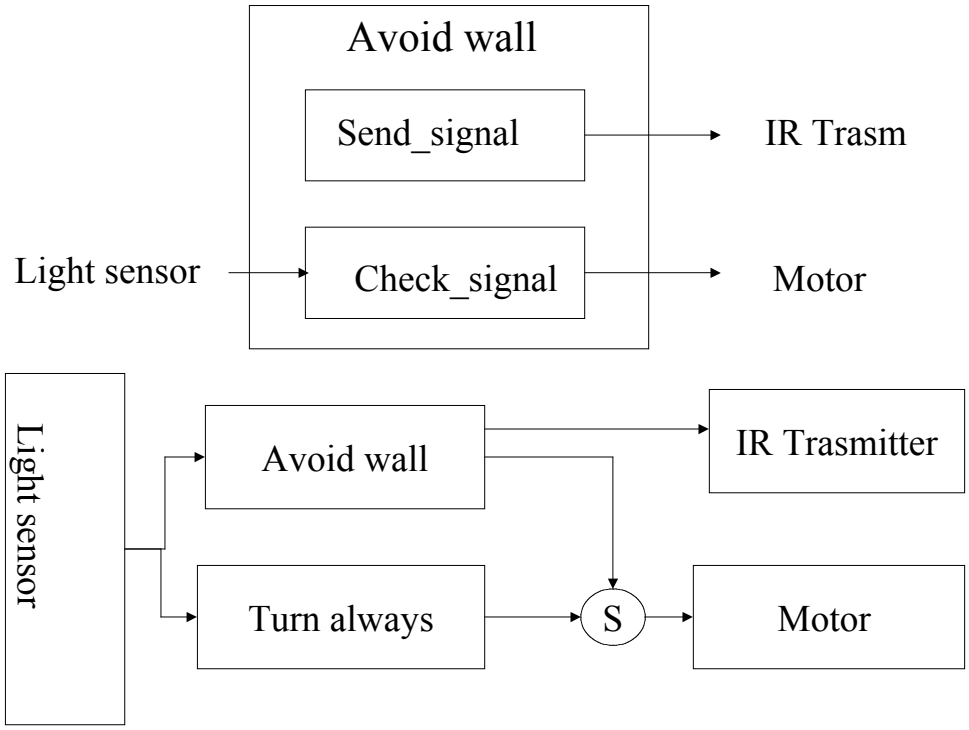


Figura 4.14 diagramma SR per *follow convex wall*

La seconda figura invece rappresenta il nostro *behavior* , il possibile conflitto tre i due viene risolto con un meccanismo di soppressione , con un coordinamento di TIPO GERARCHICO FISSO. *Turn-always* è sempre attivo mentre *avoid-wall* (anch'esso sempre attivo nel fare il check sul sensore), agisce prendendo il comando dei motori quando rileva l'IR. Una volta attivatosi mantiene il comando dei motori finché si è compiuto l'allontanamento.

NQC language

Nell'implementazione il task che implementa *Turn-always* è stato chiamato `Check_Turn`., il suo compito è quello di sterzare, avanzare, sterzare, avanzare…ogni 0.3 sec. per stare sempre in vicinanza del muro .La sterzata è stata rallentata con dei blocchi (spegnimento) dei motori per permettere al light-sensor di recepire la variazione di luminositá (dovuta all'infrarosso) altrimenti tale variazione era troppo repentina a causa dell'elevata velocitá di rotazione; in veritá'essendo un loop infinito no compi un arco id curva ad ogni iterazione ma un arco di curva seguito da un moto rettilineo per 0.8 secondi.
La soppressione vista nel diagramma SR avviene da parte del task `check_signal` che stoppa e riattiva `il` task `check-turn`

```
#defineLEFT OUT_A
#define RIGHT OUT_C
#define TIMER_TURN 3 // era a 15 per rcx frontale
#define TIME_FAIR 50
int lastlevel,sem=0;

task send_signal()
{
  while(true)
    {SendMessage(0); Wait(10);}
}

task check_signal()
{
```

```
  while(true)
  {
    lastlevel = SENSOR_2;
    if(SENSOR_2 > lastlevel + 200) {
      //OnFwd(LEFT+RIGHT);
      stop check_turn;
      PlaySound(SOUND_CLICK);
      OnFwd(RIGHT);
      Rev(LEFT);
      Wait(23);          //  30 gradi .... era a Wait(50) con rcx
frontale
      OnFwd(LEFT+RIGHT);
      ClearTimer(1);
      start check_turn;
     }
   }
}
task check_turn(){
   while(true){
     if(Timer(1)>=TIMER_TURN){
     Rev(RIGHT);
     Wait(12);              // 0.35 secondi ==>45 gradi
     Off(RIGHT);
     Wait(2);
     OnRev(RIGHT);
     Wait(12);
     Off(RIGHT);
     Wait(2);
     OnRev(RIGHT);
     Wait(12);
     //Raddoppiandolo sono sicuro che curva
     Rev(RIGHT);
     Wait(12);              // 0.35 secondi ==>45 gradi
     Off(RIGHT);
     Wait(2);
     OnRev(RIGHT);
     Wait(12);
     Off(RIGHT);
     Wait(2);
     OnRev(RIGHT);
     Wait(12);
     Fwd(RIGHT);
     Wait(80);
     ClearTimer(1);
    }
   }
 }
task main()
{
  SetSensorType(SENSOR_2, SENSOR_TYPE_LIGHT);
  SetSensorMode(SENSOR_2, SENSOR_MODE_RAW);
  SetPower(OUT_A+OUT_C,4);     //con batterie scariche  non occorre
  OnFwd(OUT_A+OUT_C);
  start send_signal;
  start check_signal;
  //start check_turn;
  ClearTimer(0);
  ClearTimer(1);
 }
```

4.5.7 Folllow-wall

Sensore Necessario: proximity sensor, bump sensor (1)
Questo *behavior* risolve i problemi del precedente riuscendo a seguire qualsiasi tipo di parete, quindi potendo entrare in qualunque stanza a prescindere dalla posizione della porta.

Utilizza i *behavior turn-always* , *avoid-wall* e *obstacle avoidance,* quest'ultimo non è l'originale ma una versione leggermente modificata in cui si sono invertiti i comandi ai motori conseguenti alla pressione dei bump. Inoltre è necessario solamente un bump sensor: quello di destra, posto nella classica posizione frontale (Figura 4.12)

Diagramma SR

Per i *behavior avoid-wall* e *obstacle avoidance* si è utilizzato il metodo di COORDINAMENTO COMPETITIVO, ovvero il primo che si avvia porta a compimento la sua opera: un task interdice l'altro e viceversa. Mentre entramb hanno un livello di priorità maggiore su *avoid-wall.* Quindi si è i presenza di 2 strategie di coordinamento una ad alto livello ed una a basso.
In Figura 4.15 sono stati riportati tutti i *behavior* , si otano distintamente i punti in cui agiscono le 2 strategie

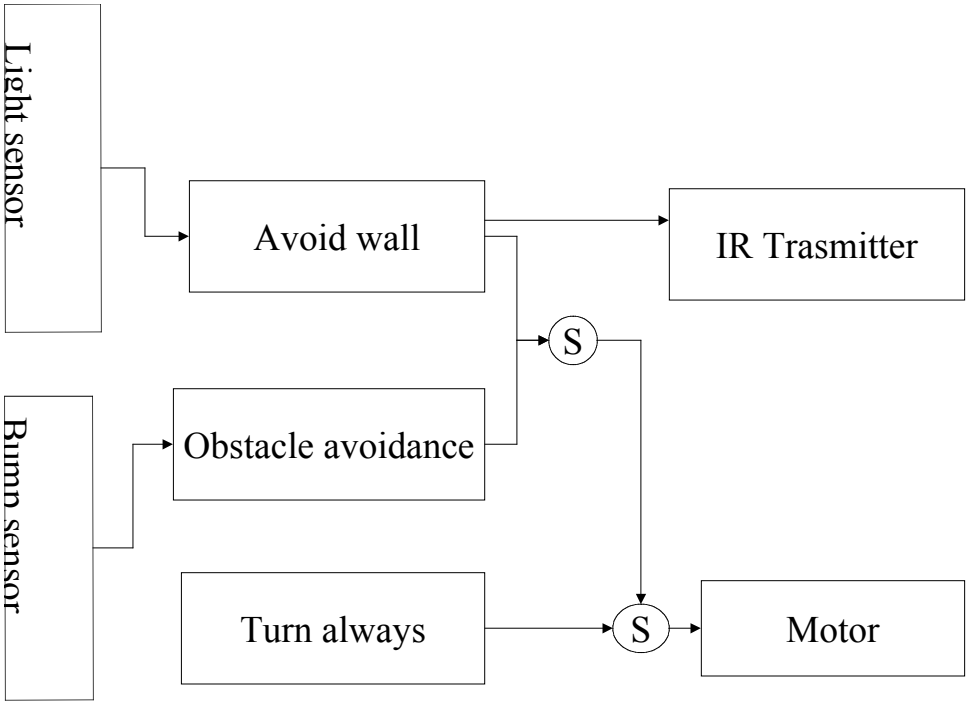


Figura 4.15 diagrama SR per *follow wall*

NQC program

Nel riportare *l'obstacle avoidance,* utilizzado solo un bump, si è utilizzata solamente la funzione `Check_left` È stato inoltre introdotto l'uso di semafori i un primo momento solo per l'accesso ai motori poi si è reso necessario estenderlo a quasi tutto il task (vedasi modifiche a *`Check-left`*).
La competizione è intesa come: il primo task che si avvia porta a compimento la sua opera ovvero un task interdice l'altro e viceversa. A noi non rimane altro che settare opportunamente le zone in cui questo può accadere.

NOTA: Non basta un check positivo sul bump per riattivare gli altri *behavior* è necessario aspettare che il tank si sia completamente spostato parallelamente alla nuova superficie e questo accade facendo lavorare il *behavior* per un certo tempo utilizzando un contatore : il `Timer(3)`, per un tempo pari a `TIMER_LEFT` inizializzata a 1 secondo.

```
/*PROBLEMA/MODIFICHE
da Competitive_behavior
```

```
P0: causa batterie nuove va troppo veloce
1.0 introdotto comando Setpower(..,4);
P1: ogni tanto il dá dei coplpetti all'indietro
1.1 in check_signal introdotto OnFwd(RIGHT)
1.2 aggiuto "On" in tutti i conamndi

2.0 tolto check_right
    check_left funziona come check_right
P1 'l'effetto del bump viene interdetto
   dal start di turn che lo fa vibrare
2.1 cut start check_turn to check_left
    cambiato radicalmente chek_left:
    immesso il comportamento di check_right
    invertendo gli Off e gli Fwd
*/
// motors
#define LEFT OUT_A
#define RIGHT OUT_C
#define TIMER_TURN 2
#define TIME_FAIR 50
// sensors
#define LBUMP SENSOR_1
#define RBUMP SENSOR_3
//timer
#define TIMER_LIMIT 50  //5 secondi
#define COUNTER_LIMIT 10  // era 20

#define TIMER_LEFT 10

int lastlevel,counter=0,sem=0;

task send_signal()
{
  while(true)
    {SendMessage(0); Wait(15);} }

task check_signal()
{
  while(true)
  {
    lastlevel = SENSOR_2;
    if(SENSOR_2 > lastlevel + 200) {
    //OnFwd(LEFT+RIGHT);
      stop check_turn;
      PlaySound(SOUND_CLICK);
      //AddToDatalog(Timer(1));
      until (sem == 0); sem = 1;
      OnFwd(RIGHT);       //==>> MODIFICA 2
      Rev(LEFT);
       Wait(23);
       OnFwd(LEFT+RIGHT);
       ClearTimer(1);
         start check_turn;
         sem=0;


      }
    }
  }

  task check_turn(){
     while(true){
       if(Timer(1)>=TIMER_TURN){
        Rev(RIGHT);
       Wait(12);             // 35==>45 gradi
        Off(RIGHT);
       Wait(2);
       OnRev(RIGHT);
       Wait(12);
```

```
       Off(RIGHT);
       Wait(2);
       OnRev(RIGHT);
       Wait(12);
       //Raddoppinadolo sono sicuro che curva
          OnRev(RIGHT);
          Wait(12);            // 35==>45 gradi
           Off(RIGHT);
          Wait(2);
          OnRev(RIGHT);
          Wait(12);
          Off(RIGHT);
          Wait(2);
          OnRev(RIGHT);
          Wait(12);
          Fwd(RIGHT);
          Wait(80);
          ClearTimer(1);
         }
        }
       }

   task watch_timer()
  {
       while(true)
       {
              until(Timer(0) > TIMER_LIMIT);
              PlayTone(440, 10);
              ClearTimer(0);
              counter = 0;
       }
  }

  void check_left()
  {
      if (LBUMP == 0)
    {
      until (sem == 0); sem = 1;
      stop check_turn;
      ClearTimer(3);
      while(Timer(3) < TIMER_LEFT)
       {
       if (LBUMP == 0)
       {
        //until (sem == 0); sem = 1;
          OnFwd(RIGHT);
       Off(LEFT);
          until(LBUMP==1);
        OnFwd(RIGHT+LEFT);
          //start check_turn;
        counter += 1;
          ClearTimer(3);
        }
       }
     sem=0;
   }
}

void check_corner()
{
       if (counter > COUNTER_LIMIT)
       {
       // we're in a corner...
       PlaySound(SOUND_DOWN);
       counter = 0;

       // back up, spin around, and continue
       until (sem == 0); sem = 1;
        OnRev(RIGHT+LEFT);
```

```
       Wait(100);
       OnFwd(RIGHT);
        Wait(50);
        OnFwd(LEFT);
        Wait(2);
        sem=0;
        }
}

task main()
{
  // configure the sensor
  SetSensorType(SENSOR_2, SENSOR_TYPE_LIGHT);
  SetSensorMode(SENSOR_2, SENSOR_MODE_RAW);
  SetSensor(LBUMP, SENSOR_TOUCH);
  SetSensor(RBUMP, SENSOR_TOUCH);
  SetPower(OUT_A+OUT_C,5);      //con batterie scariche  non
occorre
  OnFwd(OUT_A+OUT_C);
  sem=0;
  start send_signal;
  start check_signal;
  //start check_turn;
  ClearTimer(1);
  ClearTimer(0);
  start watch_timer;
     while(true)
    {
       check_left();
       //check_right();
       check_corner();
    }
}
```

4.6 Behavior che necessitano di altra sensoristica

Di seguito sono presenti alcuni *behavior* più sofisticati, necessari per poter implementare tutti i tipi di coordinamenti. Infatti si è concordato che il task generale per la prova dei coordinamenti sia del tipo:

- Partendo da un punto di una stanza
- Vai in un'altra stanza
- Prendi un oggetto
- Torna alla tana

Tutto questo senza perdersi ed evitando eventuali ostacoli nel tragitto
Come nel precedente paragrafo, per ogni *behavior* verrà indicato :

- Sensori necessari per implementarlo
- Nuove meccaniche necessarie
- Funzionamento

4.6.1 Avoid-past

Sensore Necessario: sensori di posizione
Altre necessità: funzioni trigonometriche
Un *behavior* di questo tipo permette al robot di non ripassar in luoghi in cui è già stato; spesso viene utilizzato nei *path-planning* , quando per raggiungere un goal il robot trova nella sua strada un' ostruzione e nel tentare di superarla inizia a ciclare sempre nelle stesse zone.

Un esempio, un po' preistorico di *avoid past* è il riconoscimento di angoli dell'*obstacle avoidance ,* con la differenza che quest'ultimo lavorava sul tempo e non sullo spazio, o meglio c'è la nozione implicita di autolocalizzazione, il robot è sicuro (senza averlo provato) che si trovava di fronte ad un angolo.
Per implementare l'*avoid past* è necessario una sensoristica che permetta l'autolocalizzazione del robot come un sensore di posizione, o meglio , un sensore di rotazione
Per l'autolocalizzazione è necessario poter eseguire operazioni matematiche di tipo trigonometrico quindi, non essendo l'RCX fornito di coprocessore matematico sarà necessario implementarle in qualche maniera[28].

NOTA: Si può utilizzare anche una microcamera ma l'RCX non possiede la memoria sufficiente per elaborarla, un eventuale espansione porterebbe il robot fuori della fascia di "agente a minima sensoristica"

4.6.2 Move to goal

Sensore Necessario: Sensori di posizione
Altre necessità: funzioni trigonometriche
Permette al robot, dato un target, di raggiungerlo. Target più sofisticati possono essere punti segnati in una mappa mentre più semplici possono essere le coordinate del punto da raggiungere (come nel nostro caso).
Resta sottointeso che il robot deve avere tutti i mezzi per poter raggiungere il *goal*, come i *behavior* del paragrafo precedente, a cui vanno aggiunte capacità per poter decidere se ha raggiunto il *goal* quindi capacità d autolocalizzazione: sapere da dove parte e conoscere in ogni momento dov'è.
Come per il precedente *behavior* sono necessarie anche funzioni matematiche di tipo trigonometrico, e questo è un problema non di sensoristica ma hardware e/o software

4.6.3 Forage

Sensore Necessario: Sensori di posizione
Altre necessità: funzioni trigonometriche
Permette al robot di "tornare alla tana con il cibo", quindi di andare e poi tornare al punto di partenza. Anche qui è necessario conoscere da dove parte, per poi tornarvi; metodi semplici utilizzano bei *beacon* , marcatori, per riconoscere "la tana" che nel nostro caso potrebbe essere una lampadina accesa individuabile dal sensore di luminosità [Stell 94a,c,b,c]

4.6.4 Grasping

Sensore Necessario: bump sensor, sensore di rotazione
Altre necessità: un motore in più
Questo *behavior* permette al robot di afferrare un oggetto ,le braccia semoventi sono costituite da pezzi lego, mentre il motore e degli ingranaggi permettono di chiuderle. È necessario un motore in più perché gli altri sono impegnati nella locomozione del robot.
Eventualmente se è disponibile può essere utilizzato un *bump sensor* che , posto in mezzo alle braccia , indica quando il pezzo da afferrare è in mezzo alle stesse ,ovvero finché non viene premuto , le braccia non vengono chiuse; può essere anche utilizzato per monitorare se durante il trasporto l'oggetto è caduto oppure se la morsa si sta allentando. La trasmissione della potenza meccanica da motore a braccia non avviene con ruote dentate ma con

[28] Si imanda al capitolo 7

cinghie e pulegge in tal modo a chiusura terminata il motore non rischia di bruciarsi perché la ruota è bloccata ma può continuare a ruotare, come già espresso nel Cap 2.

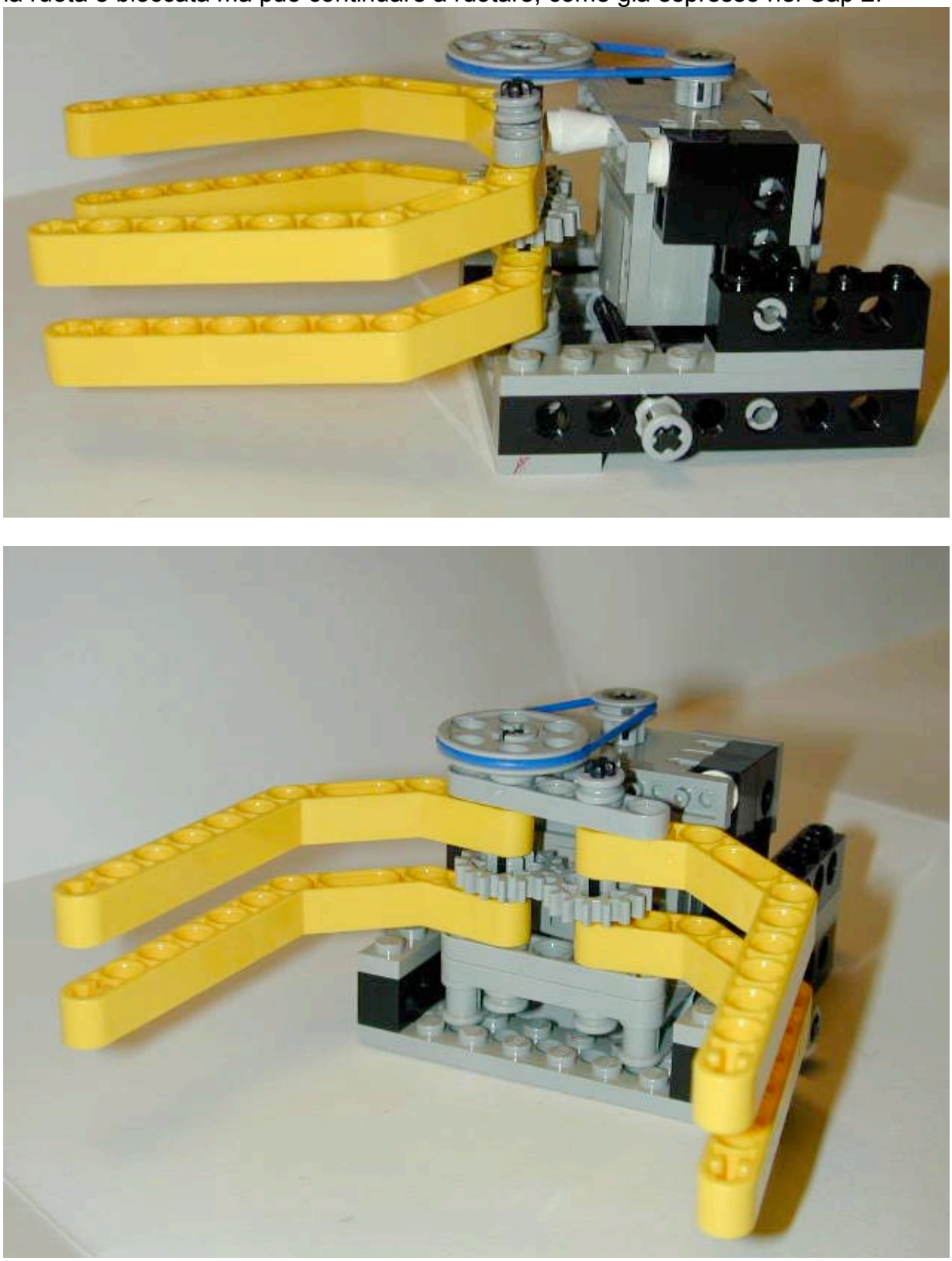


Figura 4.16 Utilizzo del terzo motore per il grasp

Capitolo 5 LegOS: Nuovo operative system

LegOS: Nuovo operative system

Per implementare il controllo della nuova sensoristica ed di un coordinamento dei behavior non elementare è necessario dotarsi di un Sistema Operativo più potente: si è scelto il LegOS , acronimo di LEGO Operative System.
Nei seguenti paragrafi ne verranno illustrate le caratteristiche, descritto il comportamento a livello di kernel e come avviene la gestione delle periferiche. La trattazione non vuol essere un manuale sul LegOS ma vuol mettere l'accento su alcuni modi di operare di questo OS[29] e su alcuni comandi utili per la programmazione, al fine di dare gli strumenti a chiunque volesse modificarne il kernel oppure sfruttarlo nella programmazione ad alto livello.
Ulteriori dettagli possono essere trovati in appendice.

5.1 Caratteristiche

La creazione del LegOS è iniziata nel Ottobre del 1998 ad opera di Markus Noga [Noga 98], ed ha continuato ad evolversi come open source project nel sito http://legOS.sourceforge.net [http2]

5.1.1 Dati tecnici

In contrasto con lo standard del Lego Group software, LegOS offre ai programmatori l'abilità di scrivere programmi per il Lego Mindstorm in C e C++, con pochissime limitazioni.
Alcune caratteristiche sono [Noga 98][Villa 00]:

- I task dell'utente sono eseguiti come codice nativo, non interpretato
- Molta memoria, non solamente 32 variabili ma la possibilità di utilizzare 32k
- Linguaggio di programmazione text-based, uso dello stesso *"gcc"* che compila Linux
- Priority-based preemptive multitasking
- Sincronizzazione reale dei processi con l'uso di semafori (standard POSIX)
- Fine controllo dell'hardware: completo controllo del dysplay LCD e supporto per IR networking
- Possibilità di memorizzare fino ad 8 programmi nell'RCX
- Possibilità di utilizzare numeri in Floating point (tipi *double* e *float* in C) senza problemi
- Presenza di un generatore di numeri casuali con reale inpredicibilità.
- Possibilità di installarlo sia su sistemi Linux che Windows

Si notato subito alcune limitazioni di tipo "softwaristico":

- Il sistema ha ancora alcuni bug
- Sono richiesti *GCC* e *binutils*
- Sono richieste conoscenze di programmazione in C
- IL LegOS essendo ancora in via di sviluppo presenta scarsissima documentazione, insufficienti anche i 4 esempi forniti con il sorgente

[29] Operative System

5.1.2 Licenza di sviluppo MOZILLA

Come si è già detto il legOS è un sistema Open Source, il suo codice sorgente è pubblico quindi ridistribuibile, manipolabile e ricompilabile. Vi sono comunque delle restrizioni da rispettare.
Tutti i sorgenti del legOS sono soggetti alla *Mozilla Public License* [http 3] (versione 1.0)

```
The contents of this file are subject to the Mozilla Public License
 Version 1.0 (the "License"); you may not use this file except in
 compliance with the License. You may obtain a copy of the License
                  at http://www.mozilla.org/MPL/

Software distributed under the License is distributed on an "AS IS"
basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See
the License for the specific language governing rights and limita-
tions under the License.
```

Tabella 5.1 Licenza MPL

Questa impone di mettere quanto riportato in tabella in ogni file del legOS che si intende modificare. Indicando l'autore che ha iniziato il progetto e gli autori dei vari, successivi contributi , quest'ultimi dovranno anche evidenziare qual'è stata la modifica da loro apportata.
La licenza non ha effetto su file totalmente creati da noi, a meno che non siamo noi stessi a volerla utilizzare per tutelare i nostri copyright permettendo comunque la divulgazione.

5.1.3 CVS: Current Version System

In questo paragrafo viene illustrato il CVS che potrebbe essere definito come un metodo per sviluppare e mantenere aggiornati ed efficienti sistemi Open Source come il legOS.
Detttagli di configurazione per il legOS si trovano in Appendice .
Ciò che segue è tratto dal libro *Open Source Divelopment With CVS* [Foegel 00]

Comunemente si fa confusione tra i due vantaggi che provoca il CVS: record keeping e collaborationI pensando che questi siano correlati , invece sono strettamente connessi:

- **Record keeping**:È necessario tenerre memoria dei cambiamenti effettuati, infatti molti utenti hanno la necessità di confrontare l'attuale stato del programma con quello precedente . Se per esempio uno sviluppatore implementa una nuova caratteristica che solo una volta ultimata si scopre essere dannosa per il programma, può sempre attraverso il CVS ottenere la versione di 3 setttimane prima oppure l'ultima pubblica release

- **Collaboration.** È necessario comprendere il meccanismo chiuso con il quale il CVS aiuta numerose persone a sviluppare lo stesso progetto. È utile che i programmatori si conoscano, sappiano cosa ognuno fa e possano comunicare in modo rapido problemi di accesso ai fle. Il CVS permette (ma non incoraggia) il *file locking* in cui solo uno sviluppatore ha il permesso di editare il file , una volta terminato , toglie il lucchetto per permettere ad altri l'accesso.
  Se il gruppo diventa troppo numeroso o sparpagliato l'utilizzo dei lock diventa un vera e propria disputa che scoraggia il lavoro .

Il CVS permette a più sviluppatori di editare simultaneamente un file e si prende carico di tenere traccia di tutte le modifiche ed eventuali conflitti. Questo processo usa il *copy-modify-merge model* nel seguente modo:

1. Lo sviluppatore A richiede una copia di lavoro al CVS . L'operazione viene spesso associata al "portar fuori un libro dalla biblioteca"
2. Lo sviluppatore A edita liberamente al sua copia. Contemporaneamente un altro programmatore potrebbe essere affaccendato con la propria copia di lavoro riferita allo "stesso libro ". Entrambi lavorano rescrivendo pagine od inserendo commenti ai margini.
3. Lo sviluppatore A completa i suoi cambiamenti e comunica le modifiche al CVS che li registra. È come comunicare alla libreria quali modifiche sono state fatte al libro e perché; la libreria provvederà ad incorporare la modifiche in una *master copy* .
4. Nel frattempo un altro sviluppatore potrebbe aver inoltrato una richiesta al CVS per vedere se la *master copy* è stata modificata di recente . Se ciò è accaduto il CVS aggiorna automaticamente al sua copia *working copy*

Quindi ogni programmatore dispone dello stesso identico progetto, decidere quando richiedere aggiornamenti o trasmetterne di propri farà parte della politica individuale di ognuno. Una strategia molto diffusa prevede di richiedere sempre l'aggiornamento prima di iniziare a lavorare e di porre i commenti solamente a modifiche ultimate e testate; operando in tal modo la *master copy* è sempre in *"runnable "state.*

Conflitti

Quando sia A che B eseguono differenti cambiamenti alla stessa area di testo commentandoli, avviene un cosiddetto "conflitto". Il CVS notifica la notizia a B non appena questo inizia a commentare i cambiamenti, dicendogli che ha riscontrato un conflitto e marcando la zona della sua copia , questa presenta 2 set di cambiamenti, disposti in modo da permettere una facile comparazione. B dovrà ordinare le sue modifiche commentandole in un altro file.
CVS avvisa solamente gli sviluppatori della presenza di un conflitto, ma è compito degli stessi parlarsi e risolvere la questione.

Aggiornamento del *master copy*

Nella terminologia ufficiale di CVS la *master copy* viene indicata come il deposito del progetto (*project's repository*), in cui il deposito è semplicemente un albero di file tenuto su un server centrale[30] che deve tener conto di tutti i cicli di check-commento-aggiornamento, come nel seguente esempio:

1. A e B sono 2 sviluppatori che richiedono al copia di lavoro nello stesso istante, supponiamo che il progetto sia al punto di partenza: tutti i file sono nello stato originale.
2. Lo sviluppatore A inizia a lavorare sulla sua copia, e commenta il suo primo *patch*.
3. B guarda la televisione.
4. A, instancabile continua a lavorare commentando anche il suo secondo lotto di cambiamenti,ora la storia del deposito contiene i file originali, seguiti dal primo lotto di cambiamenti eseguiti da A, seguiti dalle modifiche tuttora in corso.
5. B continua a fare altro.
6. Uno sviluppatore C richiede la copia di lavoro, quella che gli viene fornita riflette i primi 2 set di modifiche di A, (perché sono stati depositati prima della richiesta di C).
7. A continua e finisce di commentare la terza parte

[30] Per ulteriori dettagli si rimanda al paragrafo Repository_Administration di [Fogel 00]

8. Finalmente B inizia a lavorare sulla sua copia e comincia a commentare le prime modifiche
9. Innanzitutto CVS realizza che alcuni dei file di B hanno una data antecedente l'ultima copia presente in deposito,ed avvisa B di compiere un aggiornamento prima di commentarli. Possono verificarsi 2 situazioni
    - I file modificati da B non sono quelli modificati da A
    - Qualche file modificato da B era stato già modificato da A
10. Quando B compie l'aggiornamento, CVS mischia tutti i cambiamenti effettuati da A nelle copie locali dei file di B.
    Se qualche lavoro di A è in conflitto con le modiche non commentate di B , queste non sono semplicemente applicate alle copie di B ,ma devono essere risolti i conflitti con A prima di poter essere commentate.
11. Se lo sviluppatore C richiede un'altro aggiornamento , riceverà i nuovi cambiamenti dal deposito, quali il terzo lotto di commenti di A, ed i primi (eseguiti con successo) di B. In realtà dovrebbero essere i secondi di B, ipotizzando che i primi abbiano prodotto un conflitto.

La capacità di CVS di aggiornare i cambiamenti sta nel considerare tutti i commenti dall'inizio del progetto. In pratica il deposito CVS memorizza tutte la successive differenze: per copie di lavoro molto vecchie CVS è capace di calcolare la differenza tra la copia attuale e quella corrente nel deposito e quindi di aggiornala efficacemente. Tutto sta nel gestire opportunamente i *patch.* È perfino possibile richiedere le differenze tra due particolari stati "dell'albero del deposito" senza influenzare alcuna *working copy*.

5.2 Kernel 0.2.4

NOTA: In ogni paragrafo di questo capitolo è prefisso un simbolo:

- **[O]** Indica che il paragrafo contiene informazioni strettamente legate alla dinamica del Sistema Operativo [Nielsson 00]
- **[P]** Indica che il paragrafo contiene informazioni utili per la programmazione [Villa 00]

L'accezione di "task" di seguito riportata è da intendersi come "processo" , diverso dal significato che gli viene attribuito nella programmazione robotica ad alto livello dove assume il significato di behavior.
Il Kernel del legOS è monolitico, in quanto tutto il sorgente è compilato assieme nella stessa binary image. L'interfaccia per il programmatore è data da un linker script dinamico, che contiene tutti i simboli esportati dal Kernel, (ulteriori dettagli nella sezione "Link dinamico dello user program").
LegOS usa solamente poche funzioni della ROM : quelle relative al *power on/off* ed al *refresh* del display e alla produzione di suoni.

Viene di seguito divisa la trattazione del Kernel in 7 sezioni [Nielsson 00]: Attivazione del Kernel, Timing (temporizzazione), Task Management (gestione multitasking dei processi), Memory Management (allocazione dinamica della memoria), Interprocessor Communications (comunicazione tra i processi), Semaphore, IR Networking (comunicazioni con trasmettitori infrarossi).
Prima di passare al cuore del kernel, è opportuno far chiarezza sul procedimento con cui avviene il link dinamico dei programmi dell'utente e sulla differenza tra Kernel Image, user programs, tasks.

Link dinamico dello user program

[O] Nelle versioni precedenti di legOS lo user program doveva essere staticamente collegato con il kernel ed entrambi contenuti in un'unica binary image era quindi necessario ricompilare l'intero kernel ad ogni modifica dello user program.

In questa versione il kernel provvede al caricamento dinamico del programma (rendendo possibile il load di 8 programmi nella RAM).

Il programma dell'utente è compilato in un formato rilocabile con estensione *.lx;* per scaricarlo nell'RCX si utilizza un linker dinamico ed un loader program chiamato *dll*[31] . La directory *boot* contiene 2 file *legOS.srec* , l'immagine del kernel e *legOS.lds* un linker script generato durante la compilazione del kernel, quest'ultimo associa tutti i simboli del kernel esportati con i loro indirizzi relativi sulla RAM. *LegOS.lds* è usato anche quando si compila lo user program nella fase di link, per creare il file *.lx* . Infine durante il download nell'RCX avviene il link finale relativo alla memoria allocata per il programma.

Quando avviene il download di un programma il task `paket_consumer`, risponde alle richieste di comunicazione provenienti dall'IR tower, si prende cura di allocare memoria per il programma e lo memorizza nel *programs array* . Terminata questa attività il programma può essere avviato premendo il tasto Run.

Tutte le informazioni relative agli user program sono memorizzate in un arrray contenente 8 strutture di tipo `program_t` , in ognuna delle quali sono memorizzati gli indirizzi e dimensioni dei segmenti *.text, .data, .bss* dei relativi programmi; inoltre contengono indicazioni riguardanti la dimensione dello stack, l'indirizzo di inizio del segmento di *text*, la priorità del programma ed il numero dei bytes per il download (usato per scoprire errori durante il processo di download).

[P] Nell'attuale versione, 0.2.4 non è possibile da un user program lanciare un altro programma in quanto non è conosciuto l'indirizzo delle altre strutture `program_t`. L'unica possibilità è esportare `programs array` e la funzione `program_run` dal file *legOS.lds* operazione da compiersi in modo non statico.

Rimane valido il metodo classico in cui si seleziona il programma con il tasto *Prgm* e lo si attiva premendo il tasto *Run*.

Differenze tra Kernel Image, User Programs, Tasks

Il Kernel è un'immagine binaria compilata separatamente che esporta un set di simboli (symbol table) in un linker script chiamato *legos.lds,* ed è allocata negli indirizzi iniziali della RAM.

Gli user programs sono compilati separatamente e linkati con la kernel symbol table, creando il formato rilocabile *.lx*. Il link finale agli indirizzi assoluti avviene (una volta calcolato l'indirizzo assoluto del text segment) quando il programma è scaricato nell'RCX con il comando *dll*. La zona della RAM in cui avviene il download dei programmi è situata di seguito alla zona riservata al kernel image.

I task vengono lanciati nello user program come threads, nello spazio di indirizzamento dello user program. Poiché gli user programs sono compilati in spazi separati, non è possibile lanciare threads di altri programmi.

5.2.1 Attivazione del Kernel

[O] Il Kernel vie ne attivato quando kmain è chiamato dalla ROM. Questa funzione inizializza il Kernel prima che questo parta in singletasking o multitasking mode. Se il task manager non è incluso nel Kernel è possibile attivare solamente la modalità singletasking

[31] eseguibile presente nella directory */util,* anch'esso può essere ricompilato

mentre non è possibile il caricamento dinamico dei programmi. In seguito assumeremo che tutte le potenzialità del legOS siano usate.
Durante l'inizializzazione del Kernel 3 tasks si avviano:

- Idle task: Si avvia per primo, con il piú basso livello di priorità, diventando un "dummy" task. Questo task viene eseguito quando nessun altro task necessita del tempo della CPU, il suo ruolo è mettere la CPU nella modalità power down e lo fa implementando indefinitamente l'istruzione sleep.
- Packet_consumer: Viene avviato con il più alto livello di priorità ed e' preposto a gestire le attività della porta infrarossa.
- Key_handler: Viene avviato con lo stesso livello di priorità del Packet consumer (il più alto), ed è preposto a gestire le attività relative alla pressione dei tasti presenti nel brick dell'RCX.

Quindi, una volta avviato il task manager, il Kernel inizierà l'esecuzione "swithciando" tra i 3 task. Nuovi task saranno avviati attraverso lo user program, per ulteriori dettagli si rimanda al paragrafo *5.2.3* `Task Management`.

5.2.2 Timing

[O] Il legOS è guidato attraverso gli interrupts provenienti dal timer a 16 bit il quale è configurato per generare un interrupt ogni millisecondo. L'interrupt timer è gestito da una funzione della ROM, che chiama a turno quella puntata dal ocia_vector. Questo vettore punta alla funzione systime_handler definita nel file "*Kernel/systime.c*". La funzione systime_handler sonda tutti i vari sottosistemi a turno, attraverso la chiamata dei loro handlers. Questo significa che legOS utilizza il sistema a polling invece del sistema ad interrupt per comunicare con l'ambiente.
I sottosistemi sono gestiti nel seguente ordine:

1. Incremento del system timer;
2. Chiamata al motor handler;
3. Chiamata al sound handler;
4. Timeout check sul LNP32;
5. Chiamata al button handler;
6. Aggiornamento indicatore delle batterie;
7. Aggiornamento della visualizzazione dello stato del legOS;
8. Aggiornamento del LCD dysplay;
9. Controllo del counter timeslice del task corrente ed eventuale switch task.

Per default il timeslice è di 20ms, in pratica, invece, è compreso tra un minimo di 6ms ed un massimo di 255ms [http 1].
Se il timeslice è trascorso, la funzione `tm_switcher` viene chiamata ed esegue un context- switch, cooperando con lo scheduler.

Funzione sleep,msleep
[P] Dal punto di vista del programmatore, le funzioni relative al controllo del tempo sono la `sleep(x)` e la `msleep(x)` le quali mettono a dormire il processo rispetivamnete per *x* secondi e per *x* millisecondi. Siccome queste funzioni sono vincolate all'esatto passare del tempo e non al verificarsi di un evento il loro impiego è limitato[33], rimane molto utile nella gestione del LCD , per ottenere una visualizzazione intelligibile

[32] LegOS Network Protocol
[33] Sono state impiegate nei behavior "corner detection" ed in assenza di odometria per far curvare il tank in maniera meno rapida nel "follow convex wall"

Funzione wait_event
[P] Per applicazioni più sofisticate di attesa si può utilizzare la funzione *wait_event(function name , data)* necessaria per creare funzioni di tipo *wakeup_t* ; quest'ultima ritorna zero in condizioni normali (per es. se un tasto non è stato premuto) e un valore non nullo quando avviene la condizione che si sta aspettando (per es. il tasto desiderato è stato premuto). Se si desidera testare una funzione conviene passarla alla *wait_event* insieme ad una stringa o altri dati necessari, l'OS la attiva ogni qual volta ritorna in quel thread finché la funzione non ritorna un valore non nullo, in quel momento un valore non nullo viene anche tornato dalla *wait_event()* quest'ultimo può essere usato o ignorato se necessario.
La *wait_event* è considerata una delle più importanti funzioni dell'OS il suo utilizzo viene consgilato per monitorare i bump-sensor ,la pressione dei tasti od infine per attivare il robot su cambiamenti improvvisi di luce.

5.2.3 Task Management

[O] Con la parola task manager non si intende un singolo processo o thead, ma l'insieme delle funzioni (file *kernel/tm.c*) e funzionalità che si riferiscono all'amministrazione dei task.
Prima di discutere delle proprietà di multitasking del legOS è importante fare una chiara distinzione tra la nozione di programma e la nozione di task/processo[34] (per un approfondimento vedasi paragrafo "Differenza tra Kernel Image, User Programs, Tasks")
Nel legOS un programma è l'immagine binaria memorizzata nella RAM. Un programma in esecuzione dovrà avere al suo interno almeno un processo, che a sua volta potrà lanciare ulteriori processi tramite la funzione execi (vedasi par. …).
La definizione di processo è essere un thread in esecuzione ma, attualmente, un processo è più che un thread, in quanto gestisce la divisione dello spazio della memoria del programma che l'ha lanciato.

Un processo è descritto dal process data structure (Tabella 5.2), definito come tipo `pdata_t` nel file *include/tm.h*. Questa struttura tiene le informazioni necessarie per lo switching tra i differenti task e per organizzare la coda di priorità degli stessi; include anche:
lo stack pointer salvato
un process state
un process flag
the process priority
un puntatore al processo successivo della coda
un puntatore al parent process
un puntatore alla base (inizio) dello stack
un puntatore alla funzione che viene chiamata quando la funzione ritorna da un wait state (wake-up function)

[34] Si ricorda che queste due parole sono usate con lo stesso significato.
[34] LegOS 0.2.4

data_t structure
size_t *sp_save
pstate_t pstate
pflags_t flags
pchain_t *priority
struct_pdata_t *next
struct_pdata_t *prev
struct_pdata_t *parent
size_t *stackbase
wakeup_t (*wakeup)(wakeu_t)
wakeup_t wakeup_data

Tabella 5.2 *pdata_t* structure

Il process state ha 5 differenti valori:

- *Dead*: indica che il processo è terminato e il suo stack è stato deallocato
- *Zombie*: indica che il processo è terminato ma il suo stack non è ancora stato liberato
- *Wating*: indica che il processo è a riposo e in attesa di un evento
- *Sleeping*: indica che il processo è a riposo ma è ancora attivo
- *Running*: il processo è in esecuzione

L'informazione sullo stato del processo è usata dallo scheduler che si premura di liberare lo stack a processo terminato. In ogni caso un processo può volontariamente fermarsi ed attendere un evento chiamando la *wait_event.* I parametri per *wait_event* specificano una *wake-up function*, la quale è usata per verificare se una condizione di wait è avvenuta, quest'ultima non è verificata se la wake up function ritorna ad un valore non nullo ed il processo si riavvia. Il check sulla condizione di wait è gestito dallo scheduler, avviene quando questo incontra un processo allo stato di wait, durante la ricerca di nuovi processi da eseguire. Comunque le wake up functions non disabilitano gli interrupt, questo è il compito del *systime_handler*.
PROBLEMI: Il problema in tutto questo è che la condizione di wait viene controllata solo quando il rispettivo processo viene esaminato dallo scheduler. Può quindi accadere che tale condizione sia cambiata molte volte mentre si esaminano gli altri processi. Questo può avere serie conseguenze, specialmente nel caso in cui il processo in wait attendeva un evento esterno la cui risposta doveva avvenire in modo rapido. Per ulteriori dettagli vedasi [Christensen 00].

Task structures

[O] Ogni task ha una priorità; nell'attuale versione[35] vi sono 20 livelli di priorità che vanno da 1, il più basso, a 20, il più alto. Il numero di task che possono avere la stesa priorità è limitato.
I task sono organizzati in una struttura, *task structures* (Figura 5.1), formata in una lista ordinata sui livelli di priorità. Ogni livello di priorità corrisponde ad un determinato task, quello attualmente in uso o l'ultimo ad essere stato eseguito.
Come si è visto nella Tabella 5.2 ogni struttura *pdata_t* contiene anche un puntatore all'elemento *pchain_t* che punta alla propria priority chain ed un puntatore alla struttura *pdata_t* del parent process.

[35] LegOS 0.2.4

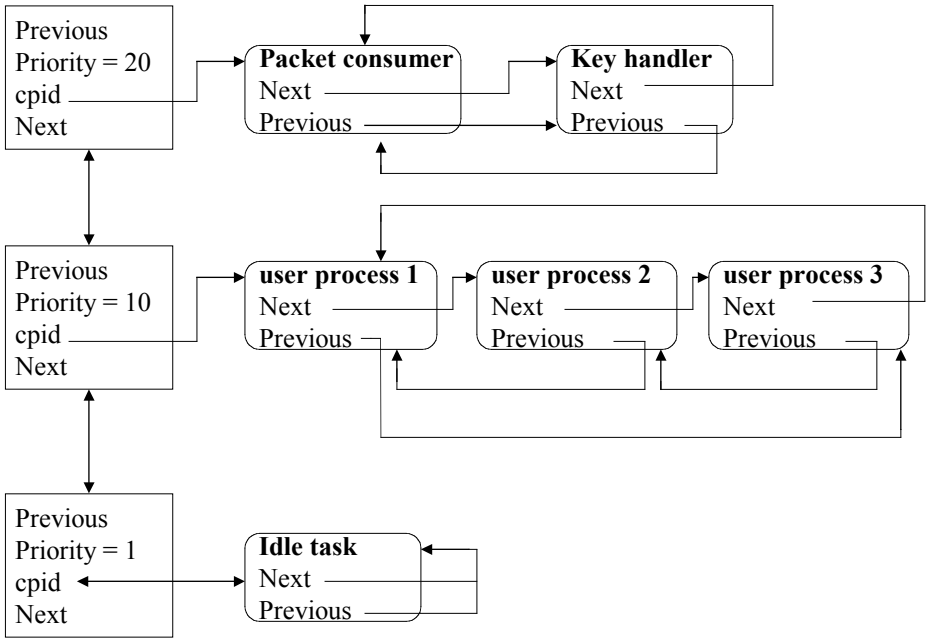


Figura 5.1 Task structures

Altri task possono essere aggiunti alla task structure attraverso la funzione *execi*, questa accede alla struttura in mutua esclusione con gli altri processi verificando un semaforo, lo stesso che si occupa di garantire che lo scheduler non stia operando mentre la task structure si sta aggiornando. L'inserzione avviene attraversando la lista dei livelli di priorità dal più basso al più alto. Una volta giunti al livello desiderato, il nuovo task viene inserito alla fine della relativa coda, se esiste, altrimenti ne verrà creata una e nella priority chaine ci sarà un nuovo livello.

Lo scheduler

[O] Quando il tempo di attività di un processo è trascorso o quando un processo volontariamente lascia la CPU , viene invocato lo scheduler per decidere quale processo sarà eseguito nel prossimo quanto temporale (*timeslice*) . Viene compiuto un `trywait` sul semaforo del task, se è bloccato significa che altri processi stanno aggiornando la *task structures* , lo scheduler allora termina ed nessun nuovo processo sarà schedulato . Se viceversa non è bloccato, lo scheduler lo bloccherà ed esaminerà lo stato del processo corrente. Nel caso sia uno *zombie,* deallocherà la sua memoria e lo rimuoverà dalla *task structure* ; nel caso sia l'ultimo processo nel livello di priorità,il livello verrà rimosso dalla *priorità level list.*

Per trovare il processo successivo da schedulare vengono esaminati i livelli di priorità in ordine di precedenza. Ad ogni livello viene testato lo stato del task successivo a quello finora eseguito, se è in *sleeping* il processo è pronto ad essere svegliato, viene quindi destinato all'attivazione settando il suo stato in *running*. Infine lo scheduler sistema il semaforo del task e ritorna lo *stak pointer* salvato del nuovo processo.

Se è in stato di *waiting* , viene testata la sua *`wake-up function`.* Se questa torna un balore non nullo, il processo viene selezionato per essere attivato , come descritto precedentemente. Se viceversa l'evento aspettato non si è verificato e l'attesa deve essere prolungata, viene esaminato il processo successivo nel livello. Se alla fine del livello non viene trovato alcun task pronto , lo scheduler procede ad esaminare il successivo *priorità level*

L'*idle task* non è mai in uno stato di *wait* ciò significa che sarà sempre eseguito finché non verrà trovato un altro processo eleggibile.
Lo scheduler quindi utilizza un gerarchia di priorità concentriche.
La combinazione tra: condivisione di risorse protette (uso di semafori) e un ordine gerarchico nello scheduler, introduce un problema di *priority invertion* in cui un processo a priorità più bassa blocca uno a priorità più alta tenendo occupata/bloccata una risorsa necessaria al processo a priorità più elevata. Nessun tentativo è stato compiuto fino ad ora per evitare il problema, possono comunque essere trovate ulteriori spiegazioni e fornite alcune soluzioni in [Christiansen Pedersen Glæsner 00]

La funzione *execi*
[P] [Villa 00] Permette di lanciare i task ed è la più importante del file *unistd.h* ; la sua sintassi prevede di inserire il nome del processo e la priorità a cui vogliamo lanciarlo (se questa suoera il 20 lo scheduler ucciderà il processo). Immediatamente ristorna un oggetto di tipo `pid_t`, che non è necessario memorizzare, ma opportuno se si desidera manipolare il task in seguito (ad esempio con un istruzione di `kill(pid_t)`.
Non vi è limite nel numero di lanci in successione della funzione *execi()* e questo può accadere in qualsiasi momento: in pratica non è necessario aspettare che il task lanciato sio terminato per poterne lanciare altri
Vi è infine la possibilità di passare alla funzione altri due parametri, `argc` e `argv`. In molte circostanze i thread solo lanciati quando il programma li chiama per la prima volta ed in tal caso si passa 0 per `argc` e NULL per `argv` . Se viceversa si desidera lanciare un thread da un'altro thread quando il programma è già partito può verificarsi la necessità di passare dati raccolti dalla prima stringa alla nuova stringa, è possibile passare i dati mettendo in `argc` il numero degli oggetti che si desidera passare ed in `argv` l'argomento
Per dettagli di programmazione si rimanda all'Appendice B

`&exit`	Frame del nuovo task
`&task code start`	
`ROM save ccr(=0)`	ROM timer interrupts
`ROM saved r6(=0)`	
`&ROM return address`	Systime handler
`callee saved r0`	
`&systime return`	
`saved r1(=argv)`	
`saved r2(=0)`	tm_switcher
`saved r3(=0)`	
`saved r4(=0)`	
`saved r5(=0)`	

Tabella 5.3 stack frame generati dal nuovo task

[O] A livello di OS `execi` alloca memoria per l'entrata di un nuovo `pdata_t` nella coda di priorità e memoria per lo stack del task. Il nuovo processo entra nella *task structures* come sleping process. L'indirizzo della funzione `execi` è posto sullo stack e sarà questo l'indirizzo di ritorno del task ed in tal modo i task finiranno sempre attraverso al chiamata del comando *exit*, il quale si occupa di librare la memoria che lo user program ha allocato.

Concludendo quando viene creato un nuovo task vengono generati 3 stack frame :

- Stack frame per la ROM timer interrupt
- Stack frame per il *systime_handler*
- Stack frame per il task manager switcher

Funzione kill

[P] Come esposto sopra, il suo compito è quello di uccidere un processo. Quando un programma termina, motori, sensori, etc.. sono posti in modalità Off dall'OS ciò non accade per i processi: una volta eseguito il comando di *kill* sarà compito dell'utente premurarsi di spegnere sensori e motori.

ATTENZIONE: Se il kill di un task (in cui si usavano i motori) determina anche la terminazione del programma, vi è la probabilità che i motori non si arrestino; l'unica alternativa a disposizione sarebbe l'estrazione delle batterie.

5.2.4 Memory management

[O] LegOS usa un continuo schema di allocazione per l'amministrazione della memoria. Purtroppo non c'è un supporto hardware avanzato nell'RCX per il managament della memoria come *paging* o *segmentation* , quindi questa diviene una risorsa importante ciò rende necessario l'impiego di uno schema di amministrazione più elevato; a tale scopo verrà di seguito utilizzato un gruppo di funzioni defnite nel file *kernel/mm.c .*

La memoria è divisa in kernel part e user program part, il kernel code ed i dati del kernel di tipo statico sono registrati nella parte bassa memoria a partire dall'indirizzo 0x8000 all'indirizzo *mm_start* (che è una variabile globale)

Il Memory manager ha "in gestione" tutti gli indirizzi che vano dal *mm_start* all'indirizzo 0x0FFFF

La memoria è divisa in blocchi, ognuno dei quali ha 4 byte di intestazione e un numero dispari di byte di dati. (Tabella 5.4)

Process ID
Size
Data field

Tabella 5.4 Blocco di memoria elementare

&mm_start FREE 0Xef30 LCD DATA
0xef50 FREE
0xffff Motor 0xf010 FREE 0xfb80 Vectors 0xfe00 FREE 0xff00 On chip register

Tabella 5.5 Allocazione inziale della memoria

Process ID :Occupa i primi 2 byte ed indica il processo che ha allocato il blocco, nel caso in cui fosse deallocato il valore sarebbe *MM_FREE* (definito a zero).

Size: Occupa i successivi 2 byte, indica lunghezza dei dati che seguono l'intestazione

La variabile globale `mm_first_free` è utilizzata come puntatore per indicare il primo blocco di memoria libero; l'unita di memoria utilizzata dal Manager internamente è 2 byte words.
Durante lo startup, la memoria è inizializzata e contiene il blocco di Tabella 5.5 mentre `mm_first_free` punta al primo blocco libero. Tale inizializzazione viene effettuata per mezzo di 2 macro *MM_BLOCK_FREE(addr)* e *MM_BLOCK_RESERVED(addr)*

Non appena la memoria viene liberata, appaiono blocchi di memoria libera adiacenti, per ridurre il numero di frammentazioni esterne è opportuno operare un "rimescolamento"
La funzione `mm_try_join` prende come parametro l'indirizzo del blocco di memoria libero e lo "fonde" con tutti i blocchi di memoria seguenti finché non viene trovato un blocco occupato. Questa operazione richiede l'uso esclusivo della memoria, viene perciò utilizzato il semaforo del Memory Manager, `mm_semaphore`, che mette in stato di attesa tutti i processi che stanno tentando di allocare memoria. Nel caso si stia lavorando in modalità multitasking è necessario tenere bloccato `l'mm_semaphore` ad un minimo perché il merge è stato fatto al tempo dell'alllocazione, in cui il semaforo era già bloccato.

Allocazione della memoria
[O] I processi del kernel e i programmi dell'utente richiedono allocazione di memoria chiamando la funzione *malloc* . Questa inizialmente aspetta il verde dal semaforo, per essere sicura che nessun processo stia manipolando la memoria,: poi cerca un blocco libero (partendo dall'indirizzo `mm_first_free`) della lunghezza necessaria. Se la ricerca non si conclude e si raggiunge il fondo della memoria è tornato un valore *null*. Viceversa *malloc* ritorna un puntatore al blocco dati del blocco di memoria allocato.
Nell'ipotesi in cui si stia operando in mulitasking mode e si sta tentando di fondere un blocco con il successivo blocco di memoria libera e la differenza tra la lunghezza del blocco di memoria e la lunghezza richiesta supera una certa soglia (intestazione + 8 words) il blocco verrà tagliato. Quando ciò avviene verrà allocato un blocco dell'esatta lunghezza richiesta ed il rimanente spazio diverrà un blocco di memoria libera.
È opportuno scegliere la soglia con attenzione, se troppo bassa si rischia di non riuscire a tenere traccia dei blocchi, se troppo alta si provoca troppa frammentazioni interna.

Liberazione della memoria
[O] Per liberare la memoria viene invocato il comando `free`, questo semplicemente marca il blocco designato come libero (se si tratta di uno indirizzo non nullo). Come detto nello scorso paragrafo verrà eseguita una fusione con il blocco di memoria libera precedente e nel caso di *Single tasking mode* il puntatore `mm_first_free` verrà aggiornato.
Quando un processo termina (comando `exit`) viene automaticamente allocata memoria libera in quanto il comando chiama l'`mm_reaper`[36]. L'`mm_reaper` agisce in 2 modi innanzitutto marca tutti i blocchi appartenuti al processo terminato, si ricorda che lo *stack memory* non appartiene al processo ma è del suo *parent process* , quindi lo stak non viene liberato da `mm_reaper`. Se il processo è definito come kernel process , i blocchi di

[36] Nella versione ufficiale 0.2.4 c'è un bug nel comando `mm_repear`, per risolverlo è necessario aggiungere la nozione di proces flag alla struttura `pdata_t` in modo da renderla `capace` di diferenziae kernel process da user program

memoria non sono marcati in quanto tuta l'allocazione dinamica della memoria viene cancellata quando l'RCX viene spento.
Il *text* e *data static segment* del programma dell'utente sono allocati dal *packet consumer task*, considerato un kernel pocess , quindi la sua memoria allocata dinamicamente non verrà cancellata.

[P] Questo schema di management (compreso l'hardware), non prevede alcun tipo di protezione della memoria, infatti qualsiasi processo può scrivere in tutta la RAM. Anche i programmi possono allocare dinamicamente la RAM e può accadere di lavorare fuori della memoria; è necessario quindi scrivere lo user program con cura onde evitare crash di sistema.

5.2.5 Interprocess communication (IPC)

[P] Tradizionalmente l'uso di IPC nei sistemi operativi significa per i processi, avere la possibilità di scambiare dati e sincronizzare azioni. Il legOS, purtroppo, provvede ad un basso livello di semafori come primitiva di sincronizzazione. Non esiste alcuna funzione che permetta il passaggio di messaggi, ma i processi sono simili a threads, in quanto la loro memoria è condivisa. Infatti in uno stesso programma,una variabile globale può essere letta e modificata da qualsiasi processo, mentre variabili definite all'interno di un singolo task, avendo una visibilità locale non possono essere modificate dagli altri. Anche se molto elementare, l'uso di variabili globali può essere considerato un IPC.
È da evidenziare che processi in differenti programmi non presentano, a livello di kernel, alcun supporto per la comunicazione di dati.

5.2.6 Semaphore

[P] I semafori sono contatori per trhead che condividono risorse. Il LegOS è provvisto di semafori che vengono impiegati nella classica definizione di semaforo di Dijkstra, il sistema di semafori implementato è il POSIX 1003.1b[37] possono essere trovati nel file *include/semaphore.h).*
Accanto alle operazioni elementari di incremento del contatore e di *wait* finché il contatore è non nullo e decremento,vi sono un non-blocking wait chiamato `sem_trywait` ed una operazione per leggere il valore del semaforo `sem_getvalue`.
L'operazione di *wait* è implementata con una funzione `wait_envent` che mette il processo in sleep mode e specifica una funzione wake_up : la `sem_event_wait,` tale funzione ha il compito di monitorare il valore del semaforo.
[O] Se un processo è bloccato su un semaforo ed il semaforo viene segnalato, il processo non si risveglia immediatamente ma avvengono le seguenti operazioni

1. il semaforo segnala il verde, ma il processo non viene svegliato
2. Il semaforo viene incrementato
3. il ciclo successivo lo scheduler esegue il check della wake_up function del processo
4. il processo viene svegliato
5. la wake_up function esegue il decremento del semaforo, che era stato incrementato da una post operation

Al punto 2 viene eseguita un'operazione in disaccordo con la definizione data nei testi di programmazione concorrente [Ben-Ari 82]
Se più processi sono bloccatisi un semaforo che segnala il verde, il primo a cui viene controllata la wake_up function sarà svegliato: non appena la wake_up funcion decrementa il

[37] Lo stesso usato da Linux; da non confondere con il Sistem V quello descritto in [Clemente Moro 99]

semaforo, solo quel processo verrà attivato.Il check non è casuale infatti lo scheduler esamina sempre prima i processi a più alta priorità; permane comunque, nel legOS, (come descritto in precedenza) un problema, di priority inversion.

5.3 Gestione periferiche

Il legOS provvede ad interfacciare vari dispositivi all'RCX, questo comprende il controllo di:

- motori;
- sensori;
- suoni;
- LCS display;
- Tasti;
- IR trasmitter[38]

Inoltre permette di implementare altre funzioni prettamente informatiche quali:

- Float point emulation
- Random generator;
- `C++` implementation.

5.3.1 Motor control

In questa sezione verrà illustrato come vengono gestite le porte di uscita (a cui vengono connessi i motori) e come vengono implementate le routine di risposta, gli *handler*.

Hardware dei motori

[O] Vi sono tre uscite per i motori, il controllo avviene scrivendo nel byte all'indirizzo `0xf000`. Lo stato dei motori è caratterizzato da una direzione e da una velocità. La direzione è "settata" dal valore dei due bit, come evidenziato in Tabella 5.6

Controllo motori	
Valore dei bit	**direzione**
01	forward
10	reverse
00	stop
11	brake

Tabella 5.6 Controllo motori

Quando lavoriamo nel *brake mode* viene bloccata la rotazione del motore mantenendolo alimentato (un uso troppo prolungato consuma le batterie molto velocemente), mentre in *stop mode* il motore può ruotare liberamente, non venendo alimentato.
La velocità è data dalla variabile `speed` (ad 8 bit[39]), e dalla variabile `sum` (8 bit), queste sono usate per il controllo PWM in frequenza. Tutti questi parametri vengono infine scritti nella memoria che si occuperà fisicamente del controllo.

```
    IRQ1 is generated by on/off button Motors are memory mapped
    Address range is f000-fb7f to control motors
    Only accesses in this range make it off-chip
    Any write to fxxx off-chip sets external motor control registers
    Setting 40=fwd, 80=rev, 00=stop, c0=float for motor 0
    Setting 04=fwd, 08=rev, 00=stop, 0c=float for motor 1
    Setting 01=fwd, 02=rev, 00=stop, 03=float for motor 2
```

[38] Attraverso LNP, come descritto al paragrafo 5.2.7.
[39] Quindi un valore compreso tra 0 e 255.

Motor handler function
[O] Il motor *handler*, `dm_handler` viene chiamato dal *system timer interrupt handler*, che implementa l'algoritmo di disegno lineare di Bresenham. I tre motori sono gestiti a turno. La variabile `sum` viene incrementata con il *motor speed setting*, se la somma raggiunge l'overflow i bit di direzione del motore verranno scritti nei due bit di controllo; in caso contrario verrà scritto zero. Quindi se un motore ha una velocità di 255 andrà in overflow in ogni istante e conseguentemente il motore sarà guidato ogni millisecondo.

Interfaccia ad alto livello
[P] Il legOS fornisce delle macro ad alto livello per gestire tutte le periferiche. I tre motori vengono indicati con `A,B,C` e corrispondono alle uscite indicate nell'RCX (A,B,C). Ogni motore (o porta di output) è controllata da due funzioni che permettono di caricare le variabili usate dai *motor handler* :

- `Motor_X_dir`, in cui `X` indica il motore che si intende comandare; in ingresso prende un argomento di tipo `MotorDirection` il quale può essere `off, fwd, rew, brake,` (evidente la corrispondenza con i dati in tabella)
- `Motor_X_speed`, prende come ingresso un argomento di valore compreso tra 0 (`MIN_SPEED`) e `255 (MAX_SPEED)`, entrambi possono essere modificati e sono definiti in *include/dmotor.h.* Una volta applicata gli RPM (giri al minuto) dei motori dovrebbero essere ragionevolmente lineari

I motori sono inizializzati attraverso la chiamata della *motor shutdown function* che purtroppo setta tutti i motori sulla direzione `off`.

NOTA: È opportuno settare i valori corretti di `MIN_SPEED` che permettano all'RCX di muoversi in qualsiasi superficie si intenda operare con il robot, la corretta inizializzazione del parametro può avvenire per tentativi: modificando la velocità ed osservando se il robot si muove od iterativamente sfruttando un sensore di rotazione.
È opportuno ricordarsi la differenza tra *brake* e *off* per salvaguardare le batterie
Infine se capita che il robot si muove nella direzione opposta a quella programmata probabilmente si potrebbe aver invertito i contatti con i quali il motore è connesso alle porte di output (A,B,C) oppure averli collegati alla porta sbagliata o nell'ipotesi peggiore che i fili si siano bruciati.

5.3.2 Sensor control (on-chip A/D)

In questo paragrafo si illustra come vengono campionati e gestiti i valori dei sensori (connessi alle porte di input) attraverso il convertitore A/D.
Il controllo può essere modificato manipolando i file *kernel/dsensor.c* e *include/dsensor.h*

On-chip A/D converter
[O] Il microcontrollore presente nell'RCX include un convertitore analogico digitale ad approssimazioni successive da 10 bit che può essere usato sugli 8 canali di ingresso analogici (`AN0-AN7`).
Il convertitore dispone di:

- un set di registri mappati memoria agli indirizzi `0xffe0-0xffe7;`
- `4` registri di dati, `ADDRA-ADDRD` da 16 bit contenenti il risultato della conversione di 4 canali di ingresso. Sono registri a sola lettura e divisi in una parte alta e una parte bassa quando vengono mappati in memoria.

Vi è infine un registro che può essere sia scritto che letto, si tratta dello *status register*, ADCSR (8 bit) mappato all'indirizzo 0xffe8, usato per controllare il convertitore.
Quando la conversione di un segnale analogico è terminata è possibile richiedere la trasmissione di un interrupt (ADI).

Inizializzazione
[O] Come illustrato precedentemente, solo i primi 4 degli 8 possibili canali sono usati nella conversione. In particolare AN0 è usato per la porta di input 3 (sull'RCX), AN1 per la porta 2, AN2 per la porta 1, AN4 per monitorare il livello delle batterie.
Il convertitore è inizializzato dalla funzione ds_init durante l'inizializzazione del Kernel; per prima cosa viene data alimentazione alle tre porte dei sensori (ogni qual volta ciò accade si usa dire che i sensori sono "attivati"), e immediatamente il puntatore ADI (A/D converter interrupt) viene indirizzato sul ds_handler. Triggering esterni del A/D vengono disabilitati e lo *status control registrer* (ADCSR) è settato. Questa inizializzazione implica un funzionamento in *singlemode operation* con un tempo di conversione di 266 stati.
Non appena il microcontrollore si avvia, con un *system clock* di 16 Mhz, ciascuno dei 4 canali viene campionato dalle 8 alle 10 volte per millisecondo; sempre se l'interrupt handler viene preso in considerazione (abilitato).

A/D Interrupt handler
[O] Compiuta l'inizializzazione il convertitore campiona periodicamente i canali e genera un interrupt la cui routine di risposta è affidata a ds_handler. Lo scopo di questa funzione è chiamare sequenzialmente le routine di risposta dei canali, dei sensori di rotazione, dell'attivazione/disattivazione dell'alimentazione dei sensori.
Il valore della variabile globale ds_channel indica il numero del canale in cui è avvenuto il campionamento e il corrispondente *bit number* in ds_activation indica se il canale necessita di alimentazione per un corretto funzionamento. Se ciò è necessario il sensore viene attivato. Molti dei sensori sono *active sensor* e quindi necessitano di alimentazione ma questa deve essere interrotta quando avviene il campionamento, si deduce che verrà ripristinata una volta terminato il processo di conversione. Tutto ciò avviene settando il bit corrispondente al numero del canale nella I/O PORT 6. I bit in ds_activation invece, sono settati attraverso la funzione *inline* ds_passive[40] e ds_active[41] definite nel file *include/dsensor.h*.
Il sensore di rotazione usa la variabile ds_rotation e la *inline function* ds_rotation_on e ds_rotation_off. Se il supporto per il sensore di rotazione è incluso nel Kernel ed il canale corrente ha un sensore di rotazione, verrà chiamato il ds_rotation_handler. I registri R0-R3 verranno salvati prima della chiamata e ripristinati al suo termine.
A questo punto viene selezionato il successivo canale da campionare e quindi gli viene sospesa l'alimentazione. Per dare l'opportunità al sensore di non avere sbalzi di tensione il sistema viene messo in stato di *busy wait* per 32 stati. Infine l'ADCSR viene aggiornato così si conclude il campionamento del nuovo canale e la ds_handeler termina.

Lego Rotation Sensor
[O] Come visto sopra il sensore di rotazione ha una sua funzione di *handlig*, la ds_rotation_handler , che viene chiamata dalla ds_handler . Le variabili che si

[40] Permette di utilizzare il sensore in modo passivo: senza alimentarlo, per esempio nel *Bump sensor*.
[41] Permette di utilizzare il sensore in modo attivo: alimentandolo, per esempio nel *Light sensor*

occupano della gestione dei motori sono array da 3 elementi dove ogni elemento corrisponde ad un canale. Il sensore di rotazione viene calibrato attraverso la variabile `ds_rotation_set` che setta il punto di partenza sull'attuale valore campionato e, attraverso una macchina a stati finiti, viene calcolato l'angolo di rotazione.
I valori quantizzati sono mappati in un range da `0x0-0xf` e corrispondono ai 16 stati; ad ognuno di essi viene associato un valore compreso tra –1 e 3. Questo numero è usato per riconoscere quando il valore dell'angolo deve essere incrementato di 1 o decrementato. I dettagli dell'algoritmo possono essere trovati nel file *kernel/dsensor.c*.

Interfaccia ad alto livello per i sensori

Poiché i valori convertiti dal A/D sono dei registri di dati mappati in memoria, possono essere letti direttamente usando delle macro: `SENSOR_1 SENSOR_2 SENSOR_3`, definiti nel file *include/dsensor.h*. Si ricorda che questi sono i valori quantizzati[42], altre macro sono definite per indicare tutti gli altri tipi di dati letti dai sensori
Il valore delle batterie viene indicato dalla macro `BUTTERY`.

Interfaccia ad alto livello per il Light sensor
[P] Le tre macro per i sensori di luminosità sono `LIGHT_1…LIGHT_3`, come si può intuire il numero 1 indica che il sensore di luminosità è posto nell'input 1 e così via. Queste macro mappano il valore quantizzato in un valore tra 0 (totale oscurità/nero) e massima luminosità/bianco.
È possibile scegliere se alimentarlo o no attraverso le funzioni:

- `ds_active` pone il sensore in modalità attiva, come input prende l'indirizzo del sensore (&SENSOR) da attivare e la porta di ingresso in cui è collegato.
  Quando si usa tale funzione si può notare l'accendersi del led rosso, utile per illuminare le superfici in cui si avvicina il sensore quando si vuole misurare la loro riflettività indipendentemente dalla luce ambientale
  Ritorna un valore compreso tra 50 e 300
- `ds_passive` si utilizza questa funzione per non alimentare il sensore quando per esempio si vuole analizzare la luce ambientale o individuare una sorgente luminosa in un ambiente buio. In questo caso il led rosso è spento
  Ritorna un valore compreso tra 220 e 280

NOTE: Nella modalità passiva il valore misurato non è stabile come nell'attivo , inoltre il valore di totale oscurità varia anche dallo stato della batteria, è quindi opportuno tararlo
Nell'attivo, se si poggia il sensore su una superficie si rischia rilevare misure più basse.
È infine opportuno ricordarsi che il responso del light sensor non è strettamente lineare[43]

Interfaccia ad alto livello per il Touch sensor
[P] In modo analogo al Light sensor sono definite le macro per i sensori di contatto `TOUCH_1…TOUCH_3`. Il valore dei *bump sensor* è mappato in un numero binario: 1 quando vengono attivati/premuti, 0 in caso contrario.
NOTA:Verificare che il sensore non balzi indentro troppo poco durante l'urto oppure se avvenuto l'urto non rimanga per qualche istante ancora premuto; il peso degli oggetti che lo stimolano deve essere opportuno, tale da rendere la pressione sul *bump sensor* elastica.

[42] Il loro valore è compreso tra 0 e 1023.
[43] Si rimanda al Capitolo 3 Sensoristica di base

Interfaccia ad alto livello per il Rotation sensor
[P]Il valore dei sensori di rotazione vengono ottenuti dalle macro `ROTATION_1…ROTATION_3` che si riferiscono all'array `ds_rotation`.
Essendo un sensore che necessita di alimentazione analogamente a quanto accade per il *Light sensor* anch'esso deve alimentato chiamando la funzione `ds_active()`. Le altre funzioni utilizzate sono:

- `ds_rotation_set`: Prende in ingresso la porta su cui è collegato il sensore ed l'angolo iniziale, viene quindi settata a questo valore la posizione iniziale del sensore . Se non viene chiamato la posizione viene settata a zero.
- `ds_rotation_on`: Attiva il processo di calcolo dell'angolo percorso.

Si ricorda che il valore tornato è aggiornato ogni 1/16 di giro, parte da quello settato e viene incrementato se ci si muove in una direzione e decrementato se si va nell'altra.

5.3.3 Sound control

In questo paragrafo viene descritto in che modo è gestito lo speaker dell'RCX , tutte le funzioni si trovano nel file *include/dsound.h*

Sound handler
[O] Il controllo delle capacità acustiche del brick è gestito dal *sound handler*, il suo ruolo è suonare le note e le pause (gestire i tempi in cui non viene prodotto alcun suono).
Una musica è una struttura di tipo `note_t` composta da 2 nodi:

- *Duration* (durata) : le lunghezze valide sono *whole* (intera)*, half* (mezza)*, quarter* (un quarto)*, eight* (un'ottava); ad una lunghezza di 1/16th corrisponde un tempo di 200 millisecondi
- *Pitch* (tono): il tono specificato viene traslato in frequenza indicizzandolo in un array chiamato `picht2freq`.

Il sound handler viene guidato dalla chiamata del *system time handler* , lavora principalmente con due variabili `dsound_next_note` e `dsound_next_time` e un array di `note_t` . La funzione di handler inizia verificando se una nota potrebbe essere suonata comparando il *system time* con il `dsound_next_time`; se il *system time* è più grande, viene suonata una internota (pausa) o una nota, a seconda del valore del *internote flag*.
L'*internote* non usa i dati del *node array* e viene suonata ogni secondo, per default ha una durata di 15 millisecondi. Infine, il `dsound_next_time` viene aggiornato e l'*internote flag* resettato.
Una volta terminato il suono di un `note_t`, il `dsound_next_time` e il `dsound_next_note` sono aggiornati e l'*internote flag* è settato in modo da indicare che la nuova nota da suonare è un internote.

Note hardware
[O] Non scenderò nei dettagli di come viene prodotto il suono, ma se si vuole utilizzare in modo ottimale lo speaker è necessario conoscere almeno i principi base, (ulteriori informazioni nel file *kernel/sensor.c*).
Una nota viene suonata attraverso la funzione inline `play_freq`, che comunica con l hardware attraverso gli I/O *memory mapped*. Viene usato il canale zero del timer on-chip da 8 bit per far suonare differenti frequenze allo speaker.Il suono viene appunto creato ponendo il segnale generato dal timer nello speaker .

La funzione play_freq prende in ingresso la frequenza desiderata a 16 bit dei quali il MSB byte[44] viene usato per decidere il valore del timer mentre l' LSB viene usato per selezionare il clock interno che funge da ingresso al timer.

Interfaccia ad alto livello
[P] Sono previste una seri di macro predefinite per indicare i 4 tipi di durate WHOLE, HALF, QUATER, EIGHT e i toni delle diverse note PITCH_C1 (DO maggiore), PITCH_Cm1 (DO#), PITCH_D1(RE), PITCH_Dm1 (RE#), PITCH_E1 (Mi)…PITCH_Gm8(La#) ,PITCH_A8 (Si), per un totale di 8scale complete.
Un processo può generare suoni creando un array di note_t e chiamando la funzione dsound_play (con l'array come parametro). In questo modo il task una volta chiamata può continuare a fare alte cose mentre il *system time interrupts* quida il *sound driver.*
La generazione di suoni viene interrotta chiamando il dsound_stop, se in un nodo dell'array viene trovato un *null*. Il dsound_stop si occupa di suonare una pausa, poi resetta le variabili di suono.
Esiste infine un array di suoni predefiniti dsound_system_sounds ma a tutt'oggi solo un suono è definito il *system BEEP*
NOTA: Un solo task per volta può utilizzare il dispositivo di generazione dei suoni.

5.3.4 LCD display control

In questo paragrafo viene descritto in che modo avviene il controllo del display LCD, e cosa è possibile visualizzare in esso. L'LCD è sorprendentemente versatile grazie a questo OS è possibile controllare totalmente il display arrivando a poter scrivere su ognuno dei 7 segmenti che compongono ognuno dei 5 *digit*. Le funzioni utilizzate si possono trovare in *include/dlcd.h, include/ROM/lcd.h, include/conio.h, kernel/conio.c, kernel/lcd.h*

LCD handler
[O] L'LCD è controllato dal *LCD handler,* guidato da un timer interruptus. La funzione di handler è lcd_refresh_next_byte che viene chiamata ogni 6ms; un totale refresh del distplay avviene ogni 54ms ovvero circa 18-19 volte al secondo
L'LCD può visualizzare 9 byte di dati, per questi vi sono 2 arrray: display_memory definito nella ROM e lcd_shadow che contiene l'ultimo byte visualizzato, viene utilizzato per verificare se vi è la necessità di un refresh ovvero se il byte attuale è diverso da precedente.

Interfaccia ad alto livello
[P] Sono previste numerose macro per la visualizzazione sul display dei valori dei sensori, batterie etc.…anche per la gestione di uno degli 8 *dots* di ogni *digit* (cifra): LCD_0_TOP, LCD_0_MID,LCD_0_BOT,LCD_0_TOPR,…,LCD_4_BOTR,LCD_4_TOPL,LCD_4_BOTL, LCD_4_DOT. Oppure si può accedere ad un intero *digit*
Può essere operato un refresh manuale chiamando la fuzione lcd_refresh che rivisualizza tutti i 9 byte contenuti in display_memory; nelle vecchie versioni 0.1.x era necessario chiamarlo , nell'attuale versione invece avviene automaticamente
Le funzioni più importanti sono:

- lcd_int scrive un intero nel buffer (visualizza un intero)
- putw visualizza word esadecimali

[44] Il byte più significativo

- `lcd_show` permette di attraverso una maschera di ingresso di visualizzare qualsiasi tipo di lettera, esiste una lista di codice nativo nella ROM che può ssere sfruttata per le visualizzazioni più comuni
- `cls()` cancella l'intero display
- `cputs` serve per visualizzare una stringa di 5 caratteri, ulteriori caratteri veranno tagliati.

Queste funzioni sono definite in termini di `cputc_native` che visualizza singoli numeri esadecimali e `putc` che visualizza singoli valori ASCII avendo accesso al codice nativo dei segmenti. Per dettagli vedasi appendice.

5.3.5 Button control

Nel legOS è possibile anche controllare i tasti presenti nell'RCX modificandone i comportamenti, come ad esempio utilizzare il "View" per uccidere dei task ed ativarne altri oppure visualizzare ad ogni pressione determinate variabili interne de programma.

Button handler

[O] Il `dkey_handler` viene chiamato dal *system time handler.* Lo stato del bottone viene verificato dopo un certo intervallo "di rimbalzo" per evitare di leggere l'avvenuta pressione ad ogni istante, si aspetta quindi che il contatto del tasto non oscilli più, questo tempo è settato a 100ms. Ogni qual volta la funzione di *handler* è chiamata viene verificato il tempo di rimbalzo, se è passato (tempo rimasto =0) il tasto sarà considerato premuto, altrimenti la funzione esce

Note hardware

[O] I tasti sono connessi alle porte di `I/O` nel seguente modo:

- *On/Off* è connesso al al bit 1 della porta 4
- *Run* è connesso al bit 2 della porta 4
- *View* è connesso al bit 6 della porta 7
- *Prgm* è connesso al bit 7 della porta 7

Il valore di questi 4 bit è invertito ed usato per formare un *bit mask*. La variabile `dkey_multi` memorizza l'ultima maschera mentre `dkey` memorizza quella attuale identificando tasti premuti; se le due variabili sono identiche `dkey` non viene aggiornata.
Altrimenti la corrente *bit mask* viene assegnata ad entrambe le variabili ed il tempo di rimbalzo rimasto viene resettato a 100. In questo modo la `dkey` tiene le informazioni su quale tasto è stato premuto, ed il valore che rappresenta è lo stesso di quello tornato da `getchar`.

Interfaccia ad alto livello

[P] Come indicato, l'interfaccia per il programmatore è la funzione `getchar`, la quale ritorna il valore/stato del tasto premuto;
Io stato dei tasti è definito dalle seguenti macro:

- `KEY_ONOFF`
- `KEY_RUN`
- `KEY_VIEW`
- `KEY_PGRM`
- `KEY_ANY`

Per ottenere il riconoscimento della presione multipla di tasti è possibile compiere operazioni di OR biti a bit .

Un metodo più raffinato è l'utilizzo degli eventi di tipo wakeup_t utilizzando la `wait_event` associata alla pressione di un tasto. Si ricorda che la `wait_event` prende in ingresso un dato (la macro del tasto) ed una funzione; quest'ultima implementa altre 2 funzioni:

- `PRESSED` verifica se lo stato del dato (il tasto) è come quello desiderato (premuto), se si torna 1, viceversa 0
- `RELEASED` verifica se lo stato del dato è associato ad un rilascio in caso affermativo torna 1 altrimenti 0

Il solo utilizzo di `PRESSED` o `RELEASED` è sconveniente in quanto necessita di un tempo di pausa per prevenire i rimbalzi meccanici e di un loop per un continuo check sulla della pressione del tasto.
NOTA:La modifica del tasto *Run* o *On/Off* potrebbe provocare l'impossibilità di fermare l'RCX se non staccando le baterie.

5.3.6 IR networking (LNP)

[O] Le informazioni che seguono provengono dal LegOS newsgroup [http1].
L'LNP, LegOS Network Protocol, viene usato dal programmatre per comunicare con la porta IR, più in generale viene impiegato per comandare il robot da un host computer.
In questa versione possono essere utilizzati 2 tipi di pacchetti LNP :

- *Integrity packet:* non contiene informazione sugli indirizzi ed è usato per distribuire i pacchetti, fa in modo che il pacchetto non sia corrotto
- *Addresing packet:* è come l'*integrity packet* con in più informazioni sugli indirizzi inclusi quello di partenza e quello di destinazione. Provvede ad un servizio UDP il quale non garantisce l'arrivo del pacchetto spedito ma garantisce l'error free

[Ash 00] LNP è incluso nel kernel ma per usarlo dal PC è necessario un pacchetto ulteriore per ora disponibile solamente per Linux.
Per comunicare con PC in cui gira Linux, viene usato un demone chiamato *lnpd.* Lnpd permette la connessione multiclients all'RCX . Esiste infine una libreria chiamata *liblnp* che permette ad applicazioni di connettersi con il demone *lnpd* remoto
[P] Per programmare con LNP non sono indispensabili conoscenze in network programming (se ci sono la programmazione risulterà più facile), gran parte del codice per l'RCX è identico a quello usato sul PC
Si è detto che l'*addressing packet* ha le stesse funzionalità del UPD di internet, se il pacchetto si smarrisce non verrà comunicato alcun messaggio ne per notificare il fatto ne per richiedere la ritrasmissione , ma se arriva viene garantita la sua integrità.
La probabilità di pacchetti persi diviene molto bassa se è chiaro l'ambiente in cui si lavora e non si presentano network collision; è bene tenere a mente sempre l'esistenza probabilità .

Sul PC l'LNP viene inizializzato chiamando la funzione `lnp_init` la quale prende in ingresso 5 parametr:

1. `tcp_hostmane`
2. `tcp_port` questi 2 rappresentano il nome dell'host e della porta del lnpd deamon a cui connettersi ; è quindi possibile usare LNP da un programma che sta girando su un computer ed avere il demone e IR tower su un altro.
3. `lnp_address` è l'indirizzo di rete per il nosto programma sull'LNP network
4. `lnp_mask` determina quale bit , tra gli 8 dell'indirizzo dell'LNP determina il network device e quale bit determina il numero della porta.
5. `Flags` può cambiare il comportamento del lnpd in certe situazioni

Raramente vengono cambiate le impostazioni di default di questa funzione, se l'inizializzazione ha successo torna zero altrimenti un valore non nullo.
Una volta effettuata l'inizializzazione, affinché il programma riceva i pacchetti, è necessario "settare" un *packet handles*, questo verrà chiamato ogni qual volta arriverà un pacchetto alla porta dell'*hander;* è necessario quindi inizializzare un *handler* per ogni porta dalla quale si suppone l'arrivo di un paccheto).
L'`addr_handler_n`[45] in ingresso richiede un puntatore ai dati del pacchetto, la dimensione dei dati (massimo 255 byte) e l'indirizzo del programma che ha spedito il pacchetto. Si provvede infine alla registrazione[46] dell'*handler* sulla porta desiderata questo permetterà di completare l'installazione dell'`addr_handle_1()`

Nell'RCX non vi è la necessità di una inizializzazione, ma può essere utile settare il range del trasmettitore IR alll'inizio del programma usando `lnp_logical_range(range)`[47], potremmo così raggiungere distanze considerevoli se necessario (a discapito delle batterie). Anche nell'RCX viene chiamata la funzione `addr_handle_1` ad ogni arrivo di pacchetto, dalla legOS interupts rutines , perciò è opportuno far svolgere all'*handler* operazioni che non provochino allocamento della memoria o task switching, ma piuttosto copiare i dati in un buffer ed usare un thread per leggerlo ad ogni aggiornamento. Se la lettura avviene quando il pacchetto è già cambiato, questo viene perso. In quanto il thread che "osservatore" lavora in multitasking o meglio in un regime di time-sharing

Non è necessario che computer e RCX abbiano lo stesso range , se si desidera che il computer comandi l'RCX , senza che questo possa rispondergli allora la IR tower sarà settata come "far" ed l'RCX come "near" , in tal caso si ricorda l'impossibilità per il computer di sapere se il suo messaggio è stato ricevuto.

Debugging
[P] Una delle applicazioni più frustranti quando si programma con il legOS è la difficoltà di fare un debug. Molto spesso si è costretti a leggere le variabili interne usando l'LCD oppure i suoni , usando LNP ed un programma su PC è possibile monitorare passo passo ciò che avviene nel nostro programma. Se ho la necessità di spedire numeri e stringhe basterà nel primo byte di trasmissione indicare un codice che mi identifichi il tipo di dati trasmesso[48]
Perndere un arrary come buffer per i dati è sicuramente più veloce che allocare e deallocare un buffer ad ogni funzione di chiamata, ma c'è comunque uno svantaggio. Se l'array è una normale variabile stack questa, alla fine occuperà molto spazio e potrebbe causare overflow, mentre se l'array viene dichiarato statico il problema non sussiste, vi è invece la necessita che solo un thread alla volta chiami la funzione in cui è stato dichiarato l'array . La soluzione risulta essere l'utilizzo di un semaforo.

NOTA: Quando si ricevono i dati trasmessi dall'RCX al PC, è necessario sapere se il nostro computer, per memorizzarli utilizza il formato *Big-endian* o *Little-endian*[49]. Nel primo caso non c'è problema, anche l'RCX usa tale formato,nel secondo si dovrà scrivere un programmino che memorizza i dati arrivati in ordine opposto.

---

[45] "n" sta per numero di porta a cui si riferisce
[46] `Con il comando lnp_addressing_set_handler (MY_PORT_1, addr_handler_1)`
[47] `range` zero se siamo vicini, 1 se distanti
[48] Attraverso un programmino in C, vedasi dettagli in Appendice
[49] Nel Big-endian il processore memorizza i byte più significativi a partire dall'indirizzo più basso e i meno significativi al più alto. Esempio: `0x1234` è memorizzato, partendo dalla locazione 100. 100 contiene `0x12` e 101 contiene `0x34`. Nel Little-endian 100 contiene `0x34` e 101 contiene `0x12`.

Se si crea un programma per trasmettere la frase *Big-endian* o *Little-endian* o altre, la funzione che trasmette le frasi non smetterà di funzionare finche tutto il pacchetto non è stato spedito. Inoltre tenere memorizzate frasi occupa spazio in memoria e toglie spazio al proprio programma, (le frasi sopra citate occupano circa 1k). Sarebbe più opportuno spedire un identificativo, lasciando al programma sul PC l'attività di cercare la frase spedita in una look-up table precedentemente creata.

5.3.7 Altre funzioni

Il legOS come ricordato permette di implementare anche funzioni prettamente informatiche quali:

- Float point emulation
- Random generator;
- `C++` implementation.
- Memory optimization

Queste sono presenti nella versione 0.2.4[50], ma la versatilità dal legOS permette a chiunque di implementarne di nuove a proprio uso e consumo senza necessariamente dover ricompilare il kerel[51]

Folating point emulation

[P] Nelle versioni precedenti era possibile lavorare solamente con numeri di tipo *int* a 16 bit, quindi valori massimi di +32768 per quelli col segno ed 65536 per quelli senza segno.
In questa versione è possibile eseguire una emulazione di *floatting point* . Tipi come *float* e *doubble* possono essere usati normalmente ed il supporto per loro sarà compilato direttamente nel proprio *.lx* file .
È necessario precisare infine che un numero *float* rispetto ad un *integer* che ritorna il medesimo risultato è circa il 50% più lungo.

Rundom number generator

[P] Sono disponibili le funzioni `random()` e `srandom(x)`. La seconda può essere chiamata in qualsiasi punto durante lo start up del nostro programma, `x` è un "seme"che permette di ottenere la sequenza di numeri desiderati (passando lo stesso seme), o di generare numeri realmente random per esempio sfruttando il `LIGHT_1` sensor.
NOTA : Per l'implementazion standard di questie 2 funzioni, non vi è un seme di default, se non viene cheamata `srandom()` almeno una volta , la `random` tornerà sempre il medesimo valore.

C++ implementation

[P] Fino ad ora la totalità degli esempi disponibili, sono stati in C. in questa versione è stato incluso un supporto per il C++ ma sfortunatamente, per il momento molto materiale è disponibile solamente in CVS[52] ,vi sono comunque alcuni esempi su some usare il C++ nella directory *demo/c++/* cui è possibile trovare anche un controllo dei sensori implementato con le classi.

Memory optimization

[P] Per tutto il sistema sono disponibili 32k di memoria , dei quali la configurazione standard del legOS ne occupa 18k che possono scendere a 14k con opportune modifiche del

[50] In questi giorni sta usando la 0.2.5
[51] Per un esempio di modifica del kernel si rimanda al Cap 6
[52] Vedasi paragrafo 5.1.2

file *boot/config.h*. Configurare correttamente il *config.h* permette di non compilare determinati blocchi del legOS che si riferiscono alla gestione di alcuni dispositivi dell'RCX, che se non utilizzati, occupano memoria inutilmente .
Questi dispositivi includono:

- L'uso di thread: Task management, Semafori e Memory management[53]
- IR communication con il computer[54]
- Gestione suoni[55]
- Gestione motori[56]
- Gestione sensori[57]
- Gestione sensori di rotazione[58].

[53] Paragrafi 5.2.3 , 5.2.6 , 5.2.4 rispettivamente
[54] Paragrafo 5.3.6
[55] Paragrafo 5.3.3
[56] Paragrafo 5.3.1
[57] Paragrafo 5.3.2
[58] Paragrafo 5.3.2

Capitolo 6 Odometria

Odometria

[Borstein Everet, Feng 96] In questo capitolo viene studiato come implementare l'odometria, partendo dall'architettura delle ruote, passando per il modello cinematico, e infine analizzando un modello più vicino alla realtà, senza trascurare gli errori di tipo sistematico e non sistematico.
Per quanto riguarda il sensore di rotazione necessario per applicarla si rimanda al prossimo capitolo, per comprendere al meglio questo capitolo sarà sufficiente sapere che molto spesso verrà utilizzata la parola encoder con lo stesso significato di "sensore di rotazione " inteso come un contagiri.

6.1 Definizione di odometria

L'odometria è il metodo di navigazione più usato per l'autolocalizzazione di robot mobili, in quanto si caratterizza per:
- una buona accuratezza
- un basso costo
- riuscire a seguire alte velocità di campionamento

Comunque l'idea fondamentale dell'odometria è l'integrazione incrementale, ad ogni passo, delle informazioni fornite dal moto; questo inevitabilmente fornisce un errore.
Gli errori più rilevanti sono quelli dovuti all'errore di orientazione il cui accumulo genera un incremento proporzionale alla distanza percorsa al robot. Tuttora l'odometria rimane la più usata per le seguenti ragioni:
- i dati possono essere fusi con le misure assolute di posizione generando una stima migliore di posizione [Cox 91; HIlingum 91; Chevanier Crowley 92; Evans 94]
- può essere accoppiata a *landmark* che mi indicano la posizione assoluta in cui il robot si trova, per migliorarne l' accuratezza: si è dimostrato che l'uso sinergico dei due rende necessario un numero minore di *landmark*
- molti algoritmi di mappamento o di riconoscimento di *landmark* [Gonzalez 92] assumono che il robot possa mantenere bene la sua posizione attraverso l'osservazione di *landmark* in un area limitata e il loro continuo riconoscimento raggiungendo tempi di processo minori e riconoscimenti più accurati [Cox 91]
- È la sola informazione sulla navigazione disponibile(il nostro caso); ovvero quando informazioni esterne sono precluse per la mancanza di visibilità ambientale, di *landmark*, oppure di sensori atti al rilevamento

6.2 Architettura ruote

Il modello tipo Tank, come descritto nel Cap II , è molto versatile in terreni accidentati ma presenta problemi di slittaggio nelle curve, non essendo univoco il punto d'appoggio dei cingoli, per tale ragioni nelle applicazioni industriali, vengono tele-operati su terreni sconnessi, ma essere guidati via radio è la nozione opposta di robot autonomo.
Per applicare l'odometria si presentano 2 soluzioni:

1. collegare al tank una ruota ausiliarie, libera di ruotare su cui applicare il sensore di rotazione

2. modificare l'architettura delle ruote , sfruttando la capacità di ricostruibilità dei lego

Il vantaggio dell'avere dei lego è poter smontare e rimontare velocemente il robot, si è scelto quindi di modificare l'architettura delle ruote

Le architetture possibili per un robot possono essere:

1. *Differential drive*: Il moto avviene per mezzo di 2 ruote guidate da motori indipendenti, possono essere presenti altre ruote libere di ruotare, non collegate ai motori.
2. *Tricycle drive:* modello a "triciclo" in cui le due ruote posteriori generano il moto, essendo connesse allo stesso motore girano alla medesima velocità nella stessa direzione; la terza ruota viene utilizzata per sterzare il veicolo
3. *Arckerman steering:* simile al precedente , a 4 ruote, in cui le anteriori permettono al veicolo di curvare
4. *Syncro drive:* provvisto di 4 ruote, che vengono alimentare alla stessa velocità, lo spostamento nell spazio viene effettuato facendole ruotare del medesimo angolo
5. *Onmidirectional drive:* esternamente simile al *differntil drive* , dove le ruote altere ad avere motori indipendenti possono ruotare in qualsiasi direzione (per far ciò sono necessari altri 2 motori.) Questo modello possside 4 gradi di libertà e può spostarsi in qualsiasi direzione.

Il migliore è senza dubbio il 5, il 4 ci è recluso dalla meccanica necessaria per implementarlo ,mentre 2 e 3 assomigliano più ad automobili con i limiti di mobilità conosciuti (meno versatili da comandare come si vedrà in seguito);. Rimane la scelta numero 1

Analizziamo più in dettaglio il *trycicle ed* il modello auto; come vantaggi hanno la non necessità di monitorare la velocità delle ruote per far avanzare in modo rettilineo il robot, basta infatti posizionare la ruota sterzante in posizione di equilibrio.

L'ideale sarebbe avere un modello capace di orientare e posizionare il robot ovunque ovvero dati i parametri x, y, θ il robot dovrebbe essere capace di raggiungere tali coordinate; per farlo dispone di 2 parametri di controllo: la distanza da percorrere e l'angolo di sterzata della ruota anteriore. Ciò significa che l'orientazione e la posizione sono accoppiate, infatti per girare basta muovere avanti od indietro.

Gli aspetti negativi di questo modello risiedono nell'impossibilità di muoversi direttamente da una posizione e/o orientazione ad un'altra; per raggiungere simultaneamente una posizione e orientazione sono necessari diversi spostamenti, a volte complessi e la presenza di ostacoli complica ulteriormente la dinamica.

[Jones 93] A titolo di esempio si consideri il problema del parcheggio parallelo in un robot dotato di *differential drive* ed uno di un *like car drive* (rispettivamente "a" e "b" in Figura 6.1)

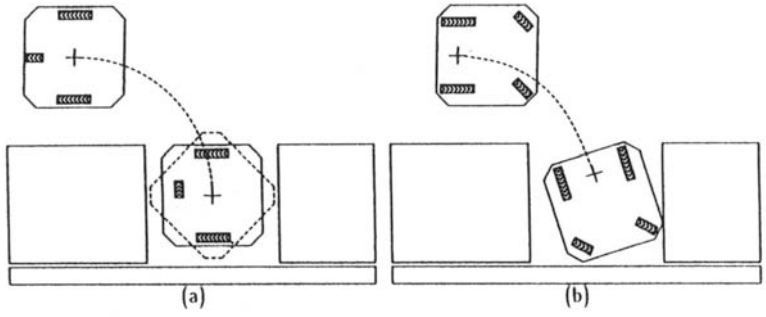


Figura 6.1 Problema del parcheggio

Si nota come il robot "a" raggiunge il *goal* avendo orientamento e posizionamento disaccoppiati: prima si posiziona poi si orienta. In "b" la stessa semplice procedura non ha effetto, sono necessari una serie di movimenti avanti ed indietro per raggiungere il *goal*

Per al versatilità e semplicità di utilizzo si è impiegato quindi un modello *differential drive* a 3 ruote, (la terza per l'equilibrio)

Posizionamento motori
Intuitivamente si può pensare di collegare le ruote motrici direttamente ai motori (Figura 6.2)
Questo rende la struttura molto stabile, con tutto il peso localizzato sulle ruote d'appoggio ma non ci permette di collegare un qualsivoglia sensore di rotazione in quanto tra la ruota ed in motore non vi è lo spazio per inserire un ingranaggio, da dove prelevare il moto e portarlo al sensore.

Figura 6.2 Configurazione ruote

Per tale ragione si è scelta la configurazione di Figura 6.3, in cui la potenza dei motori attraverso un fattore di riduzione 1:1 viene portata alle ruote da un lato ed al sensore dall'altro. Il robot ora è squilibrato verso la zona posteriore, il posizionamento del motore per il *grasping*, nella zona anteriore, è tale da riequilibrare il tutto; se si aumenta il diametro delle ruote subentra il problema "del carrello della spesa" ed ad ogni accelerazione frontale il robot si impenna.

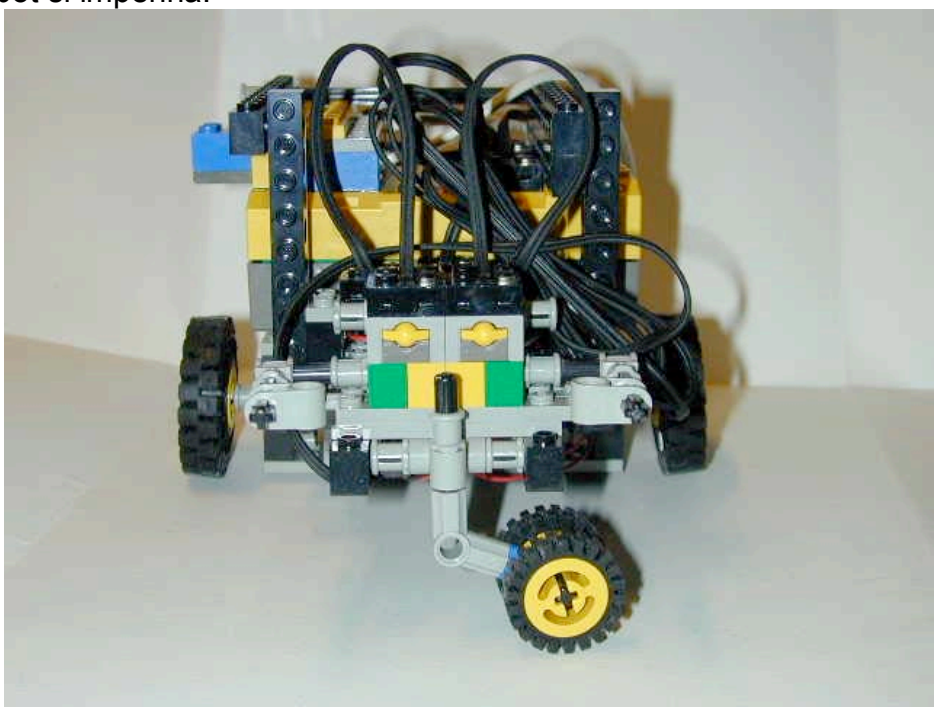


Figura 6.3 disposizione ruote

6.3 Modello cinematico

[Klarer 88; Creowley e Regignier 92] Le equazioni dell'odometria sono ricavate dall'integrazione numerica del modello cinematico (discetizzazione di Eulero) supponendo che nell'intervallo di campionamento Ts i comandi di velocità siano tenuti costanti.

MODELLO CINEMATICO

$$\begin{cases} \dot{x} = \cos\vartheta v \\ \dot{y} = \sin\vartheta v \\ \dot{\vartheta} = \omega \end{cases} \quad (6.1)$$

dove x , y sono le coordinate del robot, e θ la sua orientazione, conseguentemente ω la velocità angolare e v la velocità di avanzamento. Supponiamo che gli encoder incrementali delle ruote abbiano contato N_R per la destra e N_L per la sinistra.

Il FATTORE DI CONVERSIONE è dato da:

$$c_m = \pi \frac{D_n}{nC_e} \quad (6.2)$$

con

D_n diametro nominale delle ruote (mm)

C_e risoluzione dell'encoder pulsazioni per rivoluzione

n fattore di riduzione tra ruote (in cui è posizionato l'encoder) e quelle che guidanoil veicolo

Viene calcolata anche l'incremento di distanza compiuto dalle ruote[59] nell'intervallo esimo

$$\Delta U_{L,i} = c_m N_{L,i} \quad (6.3)$$

$$\Delta U_{R,i} = c_m N_{R,i} \quad (6.4)$$

Da questi vengono ricavate la lunghezza dell'arco spaziato dal robot ΔU_i , ed l'incremento di orientazione $\Delta\vartheta_i$

$$\Delta U_i = \frac{(\Delta U_R + \Delta U_L)}{2} \quad (6.5)$$

$$\Delta\vartheta_i = \frac{(\Delta U_R - \Delta U_L)}{b}$$

[59] Si ricorda che ogni pedice "R" si riferisce ala ruota destra ed ogni "L" alla sinistra

con *b* la distanza i punti di contatto nel terreno delle ruote.
Infine è possibile calcolare la nuova orientazione ϑ_i, e posizione del robot *x, y* all'istante esimo, nel centro di rotazione "*c*"
MODELLO CINEMATICO DISCRETIZZATO:

$$\begin{cases} x_i = x_{i-1} + \Delta U_i \cos\theta_i \\ y_i = y_{i-1} + \Delta U_i \sin\theta_i \\ \vartheta_i = \vartheta_{i-1} + \Delta\vartheta_i \end{cases} \qquad (6.6)$$

6.3.1 Implementazione.

Per ridurre gli errori di calcolo ed di approssimazione si osserva che

$$\Delta\vartheta = \frac{c_m N_R - c_m N_L}{b} = \frac{c_m (N_R - N_L)}{b}$$

$$c_m = \frac{\pi D_n}{n C_e} = \pi c^I{}_m \qquad con \quad c^I{}_m = \frac{D_n}{n C_e}$$

$$\Delta\vartheta = \pi c^I{}_m \frac{(N_R - N_L)}{b} = \pi \frac{c^I{}_m}{b} \Delta N$$

: (6.7)

$$\rightarrow \vartheta_i = \vartheta_{i-1} + \Delta\vartheta_i = .. = \pi \frac{c^I{}_m}{b} (\Delta N_1 + .. + \Delta N_i) = \pi \frac{c^I{}_m}{b} \varphi_i$$

$$\varphi_i = \varphi_{i-1} + \Delta N_i$$

Nei calcoli useremo un valore di "angolo" a meno di una costante $(\pi c^I{}_m / b)$
,utilizzando solamente le differenze sul numero di giri. Nel momento in cui servirà comunicare espressamente il valore in radianti, verrà convertito

Sfruttando il fatto che il seno con cui opero accetta solo angoli in gradi[60] e ϑ_i è per costruzione in radianti, l'argomento del seno e coseno sarà del tipo:

$$\varphi_i \cdot \pi \frac{c^I{}_m}{b} \quad [rad] \quad \rightarrow \varphi_i \cdot \cancel{\pi} \frac{c^I{}_m}{b} \frac{180}{\cancel{\pi}} = \varphi_i \cdot c_{m\deg} \quad [\circ]$$

$$con \quad c_{m\deg} = \frac{c^I{}_m 180}{b} \qquad (6.8)$$

Per quanto riguarda le posizioni *x* e *y* si può osservare che:

$$\Delta U = \frac{c_m (N_R + N_L)}{2}$$

$$x_i = x_{i-1} + \Delta U_i \cos(\vartheta_i) = x_{i-1} + \frac{c_m}{2}(N_R + N_L)_i \cos(\vartheta_i) = \qquad (6.9)$$

$$... = \frac{c_m}{2}[0 + (N_R + N_L)_1 \cos(\vartheta_1) + (N_R + N_L)_2 \cos(\vartheta_2) + .. + (N_R + N_L)_i \cos(\vartheta_i)]$$

Identici passaggi per *y,* Anche in questo caso possiamo esprimere le coordinate del robot traslate di una costante
Per motivi di fondo scala[61] infine definisco al posto di Cm/2

$$c_{mxy} = \frac{c_m}{4} \qquad (6.10)$$

eseguo tutti i calcoli con questo valore e quando dovrò memorizzare il risultato, al posto di premoltiplicare per 2 e poi eseguire lo shift, basterà *shifare* di un bit in meno[62]
Date le (6.7 ultima), (6.8), (6.10) le equazioni implementate sono:

$$\begin{cases} \varphi_i = \varphi_{i-1} + \Delta N_i \\ x_i = x_{i-1} + c_{mxy}(N_R + N_L)_i \cos(\varphi_i \cdot c_{m\deg}) \\ y_i = y_{i-1} + c_{mxy}(N_R + N_L)_i \sin(\varphi_i \cdot c_{m\deg}) \end{cases} \qquad (6.11)$$

Quando si necessità dell'angolo in radianti basterà eseguire:

[60] Per chiarimenti si rimanda a Cap 8, paragrafo 8.2 funzioni Sin e Cos
[61] Si lavora a 16 bit, le moltiplicazioni restituiscono 32 bit, i risultati vanno riportati a 16 (effettuando uno shift) . Per chiarimenti si rimanda a Cap 8, paragrafo 8.1 Fixed Point Math,
[62] Si ricorda che in base 2 moltiplicare per 2 significa compiere uno *shift* sinistro di 1

$$\vartheta_i = \varphi_i \cdot c_{m\deg} \cdot \frac{\pi}{180} \qquad (6.12)$$

6.3.2 Estratto Codice C

Nel seguito viene riportato un estratto dal file *odometry.c* in cui viene implementato il modello cinematico.
La funzione che calcola posizione e angolo si chiama *xy(),*
Se si utilizzano le ruote grandi (80 mm) Cmxy=Cm/8 questa volta sarà necessaria una moltiplicazione per 4, quindi *shift* di 2 bit

```
/*! \file   odometry.c
    \brief  Esegue il calcolo di x, y, theta in deg o rad a seconda della neces-
sitá
    \author Maragno Simone <simomail@inwind.it>
*/

#include <conio.h>
#include <include_math.c>
#include <dsensor.h>
#include <dsound.h>
/*****************************************************************************
*            pi Dn            Dn                 Cm' 180                Cm
*      Cm = ------=>    Cm' = ------      Cmdeg = -------       Cmxy = -----
*            n Ce              n Ce                   b                  8

*     Dn=  80mm  ( 42 mm ruote piccole)
*     Ce= puls/revolution =6
* n= gear raio = 6
*     b= 120 mm (dist asse ruote)          (92 ruote piccole)
* Cmdeg=3.333                                  (0x9216 2.28260 )
* Cmxy=0.87266
* (Cmxy=0.91629    0x7549     Uso Cm/4  per ruote piccole perché  Cm/2 é un nu-
mero minore di 1)
*****************************************************************************/

#define cmdeg 0xd555               //Fixed point 0.2.14
#define cmxy  0x6fb3               //s.0.15
//#define ON_C    0xffff

////////////////////////////////////////////////////////////////////////////
//
// Global Variables
//
////////////////////////////////////////////////////////////////////////////


int xy();

////////////////////////////////////////////////////////////////////////////
//                                  ODOMETRIA EQUATIONS
//
//    theta(i) = theta(i-1) + (Cm*r - Cm*l)                [rad]
//                            -------------
//                                  b
//           x(i) = x(i-1) +  (Cm*r+ Cm*l) * COS( theta )        [mm]
//                           ---------------
//                                  2
//           y(i) = y(i-1) +  (Cm*r+ Cm*l) * SIN( theta )        [mm]
//                           ---------------
```

```
//                                    2
//  Per i calcoli tenendo conto che la funz. seno lavora n gradi abbiamo imple-
mentato:
//
//                 phi(i) = phi(i-1) + (r - l)
//          phi deg= ph(i)*cmdeg                                          [deg]
//
//
//
// x(i)= x(i-1) + (r + l)* 2*2* cmxy* COS(phi deg)
//
////////////////////////////////////////////////////////////////////////////
int xy(){
      int r,l;
  static int phi=0;
      long tmp;
      static long x=0,y=0,theta=0;
      /*read_h(r_h);
      r=r_h;
      read_h(l_h);
      l=l_h;
      r_h=0;
      l_h=0;
      COUNT_STATE=ON_C; */
      l=ROTATION_1;
      r=ROTATION_3;
      ROTATION_1=0;
      ROTATION_3=0;
      phi+=r-l;
      tmp=cos((phi*cmdeg)>>14);     //[s.15.0 * 0.2.14]>>14
      tmp*=cmxy;
      //tmp+=0x2000;          //arrotondo primo bit   per  ruote piccole)
      //tmp=tmp>>14;          // eseguo shift al posto di molt per 2 e poi >>15
      tmp+=0x1000;            //arrotondo primo bit   (per ruote grandi)
      tmp=tmp>>13;            // al posto di molt per 2*2 e >>15
      tmp*=(r+l);
      x+=tmp;
      tmp=sin(phi*cmdeg);
      tmp*=cmxy;
      tmp+=0x2000;            //arrotondo primo bit
      tmp=tmp>>14;            // al posto di molt per 2 e >>15
      tmp*=(r+l);
      y+=tmp;
      return 0;
  // trasform in radianti
  /*theta*=cmdeg;
  theta=theta>>14;
  theta*=pi/180  */
  }
```

6.4 Errori

Come visto nei precedenti paragrafi l'odometria è basata su semplici equazioni che utilizzano i dati provenienti da dagli encoder incrementali. Spesso si assume l'ipotesi che il movimento di rivoluzione delle ruote letto attraverso l'encoder possa essere traslato in un moto lineare sulla superficie d'appoggio ma in realtà questo è valido solo in parte. Un esempio estremo è lo slittamento delle ruote, infatti se una ruota scivola per la presenza di

olio, l'encoder registra il movimento della ruota ma il valore letto non corrisponde ad un moto.
A parte questo esempio estremo esistono altre situazioni che provocano lo slittamento parziale e conseguentemente un inaccuratezza tra il valore dell'encoder e il moto lineare.
Gli errori si possono dividere in Sistematici e Non sistematici e sono generati da:

Sistematici
- Diametro delle ruote non uguale
- Il valore nominale del diametro delle ruote differisce da quello reale
- La larghezza delle ruote differisce da quella nominale
- Disallineamento delle ruote
- Risoluzione dell'encoder
- *Sampling rate* dell'encoder

Non-Systematic Error
- Pavimentazione sconnessa
- Oggetti sul pavimento
- Scivolamento delle ruote dovuto da:
  - Pavimento scivoloso
  - Troppa accelerazione
  - Sterzata troppo rapida
  - Interazione con forze esterne, (corpi esterni)
  - Interazione con forze interne, (orientazione delle ruote)
  - Punti di contatto delle ruote sul terreno mancanti

6.4.1 Errori sistematici

Riguardano la fisica e l'hardware/software del robot, quest'ultimi o sono dovuti distrazioni, od alla limitata conoscenza dei dispositivi on cui opero come il tempo di campionamento, la risoluzione degli A/D od infine a guasti tecnici come rottura di schede dissaldaggio di componenti etc… La loro risoluzione sta quindi in uno studio attento e preciso che mi permetta di verificare se quanto ho implementato sulle carta ha una corrispondenza nella realtà.
Gli errori sistematici dominanti riguardano:

- Diametro delle ruote non uguale:

$$E_d = \frac{D_R}{D_L} \qquad (6.13)$$

  Dove D_R e D_L sono i diametri rispettivamente destro e sinistro attuali.

- Differenza tra la distanza tra le ruote effettiva e nominale:

$$E_b = \frac{b_{actual}}{b_{nominal}} \qquad (6.14)$$

  Dove *b* è la distanza tra le ruote.

[Borsteing Feng 95] University of Michigan Benchmark. È un test in cui il robot si muove su un percorso quadrato bidirezionale. Percorre sia in senso orario che antiorario il *path* consente di evitare mutue compensazioni dei due tipi di errori che , si dimostra, avvengono spesso se si naviga in una sola direzione.

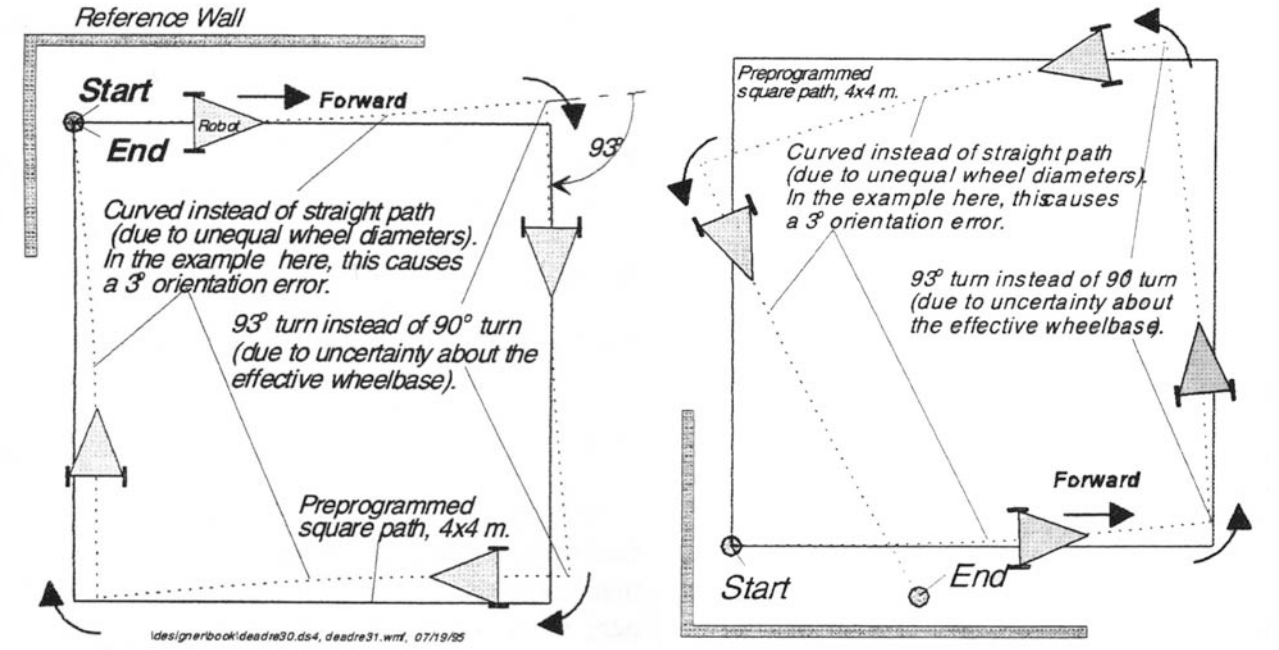


Figura 6.4 Doppio percorso

Come si nota dagli esempi in Figura 6.4 i risultati non sono identici, nel primo caso si può osservare come il manifestarsi della compensazione sopra citata: la deviazione di 3° prodotta sui rettilinei viene annullata da una sovrarotazione di 93° invece di 90°.
Il secondo esempio mostra che cambiando il verso di rotazione i due errori si somano generando sia un errore di posizione (punto di partenza diverso dal punto di arrivo) sia un errore di orientazione.

In questo test gli errori sistematici di orientazione $\varepsilon_{\vartheta_{cw}}^{sys}, \varepsilon_{\vartheta_{ccw}}^{sys}$ dove:

sys: sistemetico (*systematic*)
cw: orario (*clockwise*)
ccw : antiorario (*counterclockwise*)

per la loro natura, vengono direttamente "traslati" negli errori di posizione , che vengono calcolati misurando la distanza tra il punto d'arrivo e di partenza, nel caso di ripetute misurazioni si attua la media:

$$x_{cw/ccw} = \frac{1}{n}\sum_{i=1}^{n} \varepsilon x_{i,cw/ccw}$$

$$y_{cw/ccw} = \frac{1}{n}\sum_{i=1}^{n} \varepsilon y_{yi,cw/ccw} \qquad (6.15)$$

con
n : numero delle misurazioni

$\varepsilon x, \varepsilon y$: singoli errori di posizione
si calcola infine la vera e propria distanza sia per il percorso orario che antiorario:

$$r_{cw} = \sqrt{x_{cw}^2 + y_{cw}^2}$$
$$r_{ccw} = \sqrt{x_{ccw}^2 + y_{ccw}^2} \qquad (6.16)$$

ed infine la *misura dell'accuratezza odometrica dell'errore sistematico*

$$E_{\max,syst} = \max(r_{cw}, r_{ccw}) \qquad (6.17)$$

Non viene effettuata la media perché nelle applicazioni pratiche è opportuno considerare il più alto errore di possibile

6.4.2 Errori Non Sistematici

Sono generati dall'interazione del robot con l'ambiente, molto spesso sono accidentali ed imprevedibili: un urto che blocca il robot su un ostacolo o lo fa rimbalzare più in la, terreno liscio (scivoloso), oppure sconnesso (come può essere un tappeto in una abitazione), presenza di ghiaino, etc…. Per tale imprevedibilità non è possibile stimarli e possono assumere valori trascurabili oppure talmente elevati da compromettere la navigazione.
Le uniche misure possibili possono essere fatte su un determinato tipo d ambiente e gli aggiustamenti saranno efficaci solo in quel contesto.

Extended UMBmark

[Borenstein e Feng 94] propongono un test per valutare la suscettibilità di un robot a compiere errori di tipo non sistematico,provocando volutamente disturbi .
Viene usato lo stesso percorso quadrato bidirezionale del UMBmask , con l'introduzione di cunette artificiali costituite da 10 cavi disposti in successione di larghezza da 9 a 10 mm; la ripetizione viene attuata per fornire un elevata ripetibilità sperimentale del test ed evitare mutue compensazioni degli errori. Le due sequenze di fili vengono introdotte sul primo lato del quadrato, poste in maniera da trovarsi sotto una sola ruota del robot (quella interna al quadrato) .

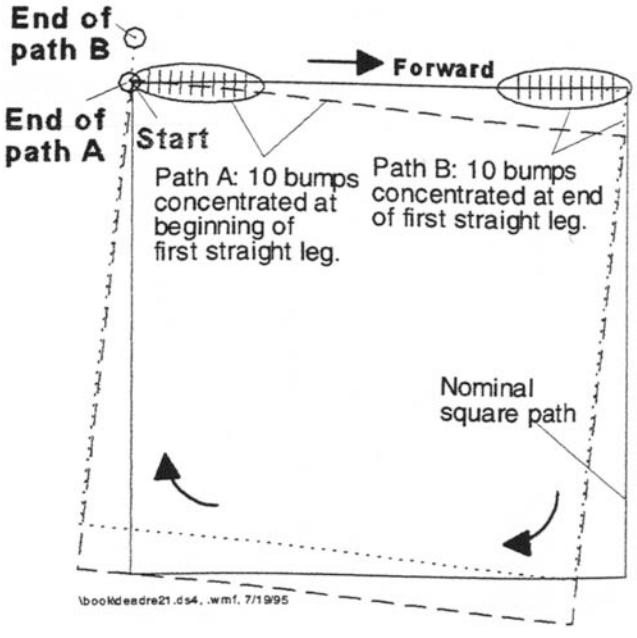


Figura 6.5 UMBmark

A seconda del percorso intrapreso , *path A* o *path B* gli ostacoli si troveranno verranno disposti all'inizio od al termine del lato.

Come si osserva in figura l'errore del percorso B è molto maggiore di quello del percorso A ma questo dato non ci fornisce uno strumento valido per capire la suscettibilità del robot a compiere errori non sistematici; l'errore di orientazione invece lo permette, analizzando *l'errore assoluto medio di orientazione*:

$$\varepsilon_{\vartheta_{avrg}}^{nonsys} = \frac{1}{n}\sum_{i=1}^{n} | \varepsilon_{\theta i,cw}^{nosys} - \varepsilon_{\theta \mathrm{a}vrg,cw}^{sys} | + \frac{1}{n}\sum_{i=1}^{n} | \varepsilon_{\theta i,ccw}^{nosys} - \varepsilon_{\theta \mathrm{a}vrg,ccw}^{sys} | \qquad (6.18)$$

con:

$$\varepsilon_{\vartheta \mathrm{a}vrg,cw}^{sys} = \frac{1}{n}\sum_{i=1}^{n} \varepsilon_{\theta i,cw}^{sys} \qquad (6.19)$$

$$\varepsilon_{\vartheta \mathrm{a}vrg,ccw}^{sys} = \frac{1}{n}\sum_{i=1}^{n} \varepsilon_{\theta i,ccw}^{sys} \qquad (6.20)$$

dove le sigle significano:
avrg: media (*average)*
sys: sistemetico (*systematic*)
nonsys: non sistematico (*non systematic*)
cw: orario (*clockwise*)
ccw : antiorario (*counterclockwise*)

Vengono sommate le medie dei valori assoluti dei contributi per evitare il caso in cui due rilevamenti forniscano angolazioni uguali ma di segno opposto che si elidano.
NOTA: questo metodo viene usato principalmente per confrontare robot diversi.
Lo svantaggio principale è la necessità di disporre di strumenti accurati per effettuare la misurazione dell'angolo

6.4.3 Riduzione degli Odometry error

L'accuratezza dell'odometria nelle applicazioni commerciali dipende dai gradi di liberta, dalla cinematica e dalle dimensioni del robot. Per quanto riguarda le ruote si è riscontrato che:

- Veicoli con ruote piccole siano più soggetti ad errori di orientazione di veicoli con ruote larghe; per esempio [Trivedi 94] con ruote relativamente piccole (340 mm), diventa necessario un reset dell'odometria ogni 10metri.
- Veicoli con le ruote il cui peso è una significante percentuale del totale, inducono un ribaltamento del robot quando invertono la direzione ("affetto carrello della spesa " che si "impenna")

Ruote leggere rispetto al veicolo non presentano tale problema [Borostein Koren 85]

- Le ruote ideali dovrebbero essere fine, con il profilo di una lama, nella pratica sono costituite di alluminio con un sottile bordo di gomma per migliorare la trazione. Questa realizzazione non sempre è possibile in quanto spesso l'odometria viene implementata sulle ruote portanti, che notoriamente devono essere larghe per consentire un maggior contatto e conseguentemente una maggior trazione.
- I veicoli che provvedono a fornire la migliore accuratezza nell'odometria sono i *sincro dirve*[63] specialmente se utilizzati in terreni sconnessi, quando una ruota è bloccata vi saranno le altre due che provvedono alla trazione e la misura dell'odometria non viene mai falsata.

Riduzione degli errori sistematici

Accorgimenti per diminuire gli errori consistono in:

- Diminuire la velocità durante le sterzate e limitare le accelerazioni
- Posizionare l'encoder non sulle ruote motrici, ma su altre ruote scollegate dai motori; per evitare che scivolamenti vengano interpretati come moto.

Systematic calibration

Gli errori sistematici fanno parte del sistema robot ed il loro periodo di variazione è estremamente lungo [Tsumura 81] quindi è possibile apportare una calibrazione univoca e duratura nel tempo. Un procedimento non difficoltoso impiega l'UMBmark [Borenstein, Feng 95a,95b] e definisce due tipi di errori:

TIPO A : è un errore di orientazione che riduce (o incrementa) l'angolo spaziato dal robot durante il percorso quadrato in entrambe e direzioni *cw* e *ccw*

TIPO B: è un errore di orientazione che riduce (o incrementa) l'angolo spaziato dal robot quando il percorso avviene in una direzione ma lo incrementa (o riduce) nell'altra.

In Figura 6.6 "a" viene mostrato un caso in cui il robot compie 4 curve per un totale di 90° ognuna ma siccome il diametro della ruota è maggiore del valore nominale,il veicolo sterza solamente di 85° per un totale di $\theta_{total} = 4 \times 85^\circ = 340^\circ$ invece dei desiderati 360°. Sia nel percorso orario che nell'antiorario la curva è minore dell'angolo desiderato:

$$|\theta_{total,cw}| < |\theta_{nominal}| \text{ e } |\theta_{total,ccw}| < |\theta_{nominal}| \qquad (6.21)$$

In "b" si nota l'errore di traiettoria nel percorrere i lati rettilinei, questo va a sommarsi all'angolazione totale nel percorso antiorario, mentre si sottrae in quello orario:

$$|\theta_{total,ccw}| > |\theta_{nominal}| \text{ ma } |\theta_{total,cw}| < |\theta_{nominal}| \qquad (6.22)$$

Il problema nasce nel distinguere i due errori e quali modifiche apportare per ridurli.

[63] È una configurazione da 3 o 4 ruote che viaggiano alla stessa velocità, lo spostamento nello spazio viene generato facendole ruotare tutte di uno stesso angolo

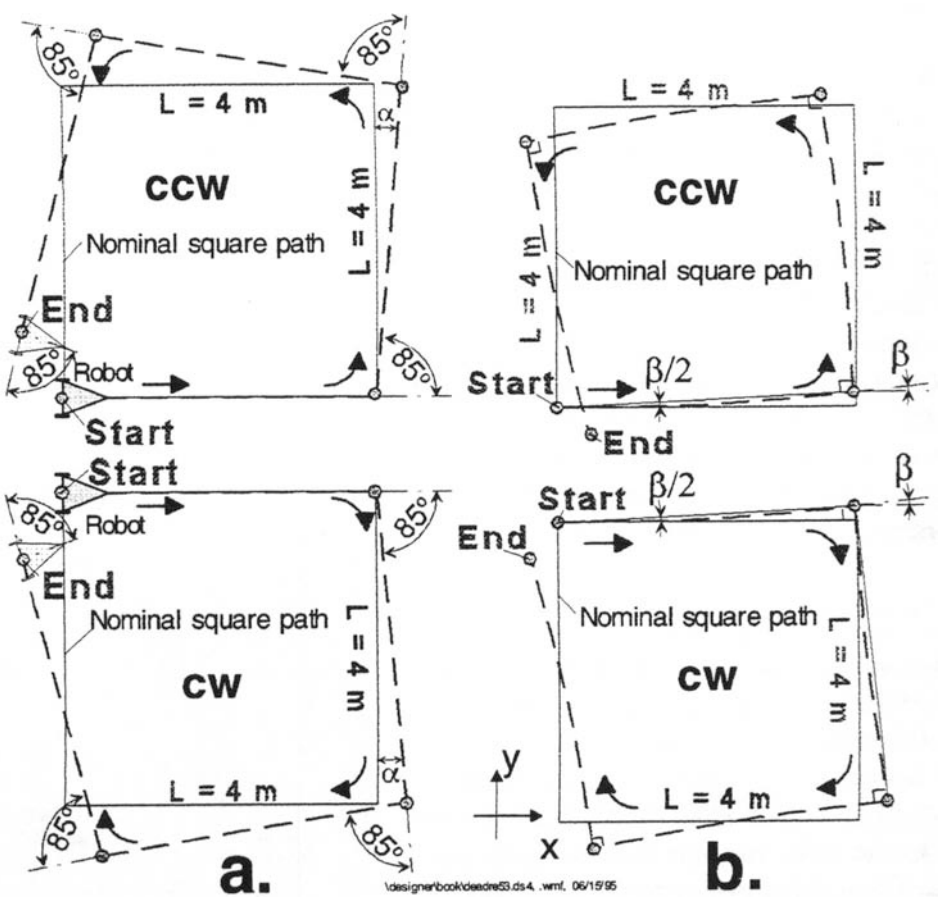


Figura 6.6 Systematic calibraion

Il Tipo A è generato maggiormente da E_b, chiamiamo le rotazioni errate α e le misuriamo in radianti.
Il Tipo B è maggiormente causato da E_d ,l'angolo di deviazione dai percorsi rettilinei lo chiamiamo β e lo misuriamo alla fine dei lati (in figura si nota che in mezzo vale metà).
Questi 2 angoli si ricavano dalle equazioni seguenti:

$$\alpha = \frac{x_{cw} + x_{ccw}}{-4L}\frac{180^\circ}{\pi} \qquad (6.23)$$

$$\beta = \frac{x_{cw} - x_{ccw}}{-4L}\frac{180^\circ}{\pi}$$

Entrambi misurati in [°]; usando semplici relazioni geometriche si ricava il raggio di curvature *R* , da questo l'errore sui diametri delle ruote e l'errore sulla distanza tra le ruote

$$R = \frac{L/2}{\sin \beta/2}$$

$$E_d = \frac{D_R}{D_L} = \frac{R + b/2}{R - b/2} \qquad (6.24)$$

$$b_{actual} = \frac{90^\circ}{90^\circ - \alpha} b_{no\min al} = E_b b_{no\min al}$$

In Figura 6.7 si possono notare gli andamenti prima e dopo la correzione

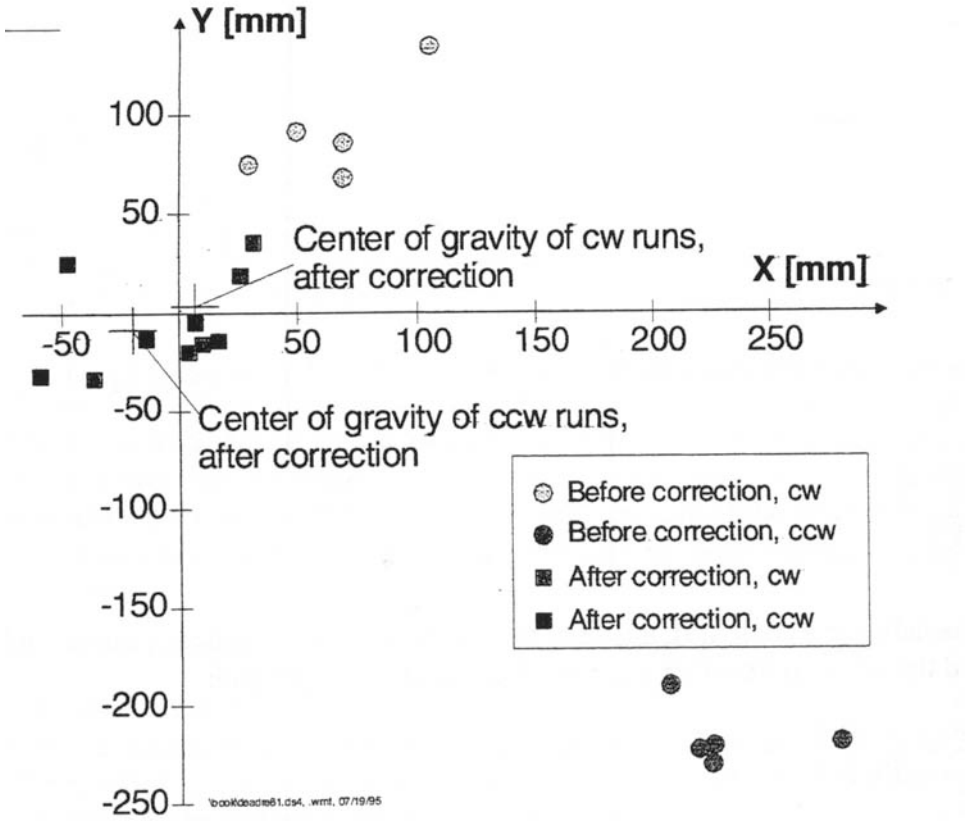


Figura 6.7 prima e dopo la correzione

Riduzione errori non sistematici

Per ridurre gli errori non sistematici i metodi migliori prevedono l'impiego di strumenti sofisticati come accelerometri, giroscopi od addirittura GPS contraddicendo l'ipotesi di minima sensoristica; oppure [Borenstein 94a] impiegando 2 robot che misurino continuamente la loro distanza relativa e con questa aggiornino la posizione riscontrata attraverso l'odometria interna, ma disponendo di un unico robot anche questo metodo è stato scartato. Ci si affida a lavorare in ambienti controllati in cui i rischi di errori non sistematici sono trascurabili rispetto a quelli sistematici.

6.5 Modello corretto

Il modello che più si avvicina alla realtà è quello di Runge-Kutta del 2° ordine esprimibile come:

$$\begin{cases} x_{i+1} = x_i + \Delta U_i \cos\left(\theta_i + \dfrac{\Delta\theta_i}{2}\right) \\ y_{i+1} = y_i + \Delta U_i \sin\left(\theta_i + \dfrac{\Delta\theta_i}{2}\right) \\ \vartheta_{i+1} = \vartheta_i + \Delta\vartheta_i \end{cases} \qquad (6.25)$$

La notazione qui utilizzata differisce leggermente dalla precedente, sono comunque di chiara interpretazione.
L'unica differenza tra i modelli riguarda l'argomento del seno e del coseno, in cui è stato aggiunto il termine $\frac{\Delta\theta_i}{2}$, le implicazioni di questo si possono notare in figura:

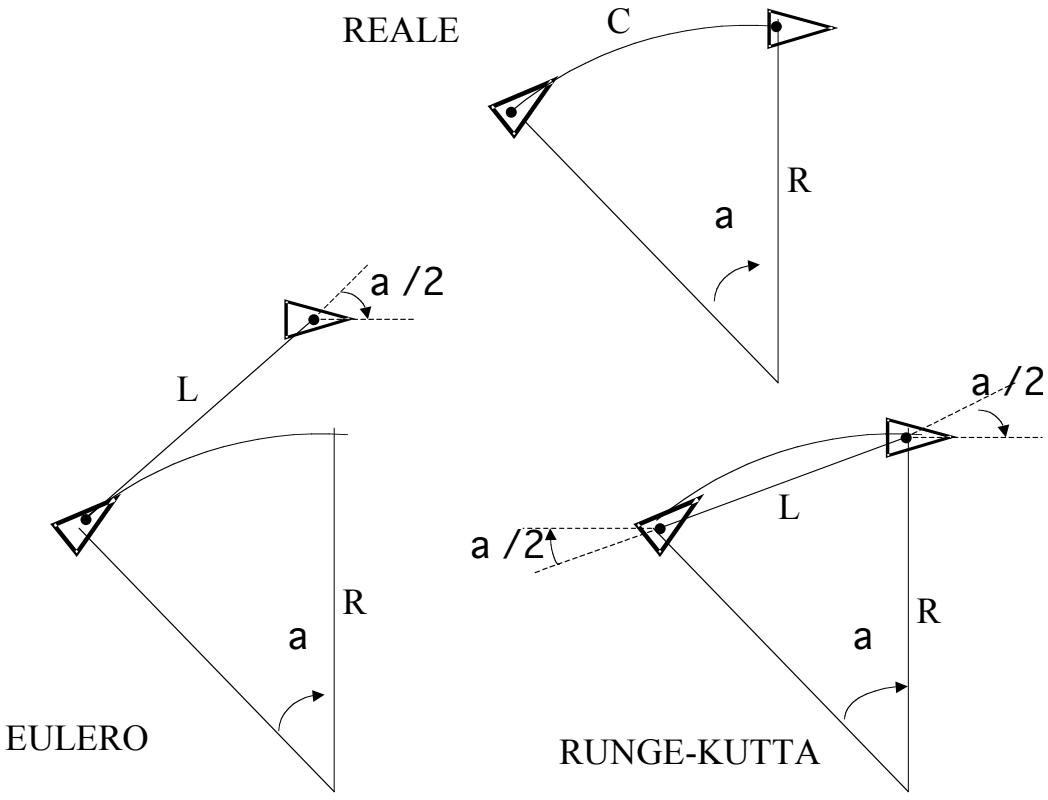


Figura 6.8 Confronto modelli

$$
\begin{aligned}
R &= \frac{\nu}{\omega} \\
\alpha &= \omega T_s \qquad (= \Delta\theta) \\
C &= \alpha R = \nu T_s \\
L &= \nu T_s \qquad (= C)
\end{aligned}
\tag{6.26}
$$

Come è possibile osservare l'errore sia di orientazione ma sopratutto di posizione è nettamente inferiore con il metodo di Runge-kutta del II ordine.

Capitolo 7 Sensore di rotazione

Sensore di rotazione

Dal capitolo 6 si è evidenziata la necessità di utilizzare un sensore di rotazione.In questo capitolo viene dapprima mostrato il sensore di rotazione fornito dalla lego, (introvabile in tutta Europa), successivamente si viene studiato come implementare un nuovo sensore di rotazione , partendo dall'interfaccia con le porte d'ingresso, passando per l'architettura hardware e meccanica ed infine soffermandosi sulle modifiche software sul kernel del legOS

7.1 Lego Rotation sensor

La lettura sul sensore di rotazione fornito dalla lego avviene in modo analogo a quanto accade per il *light sensor* : circa 8volt sono applicati per 3ms , poi viene letto il valore applicando 5volt attraverso una resistenza di Pull-Up per 0.1ms.

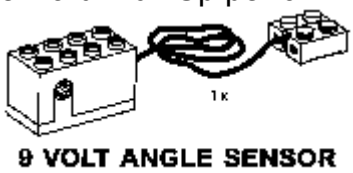


Figura 7.1 sensore di rotazione lego

Il sensore restituisce solamente 4 valori analogici: `1.8volt , 2.6volt , 3.8volt 5.1volt`.in Figura 7.2 viene mostratala sequenza con cui questi valori vengono incrementati , ogni valore corrisponde a 22.5° di rotazione quindi un giro completo avviene in 16 conteggi .
Attraverso questo modo di incrementare il voltaggio è possibile riconoscere la direzione di rotazione in ogni momento.

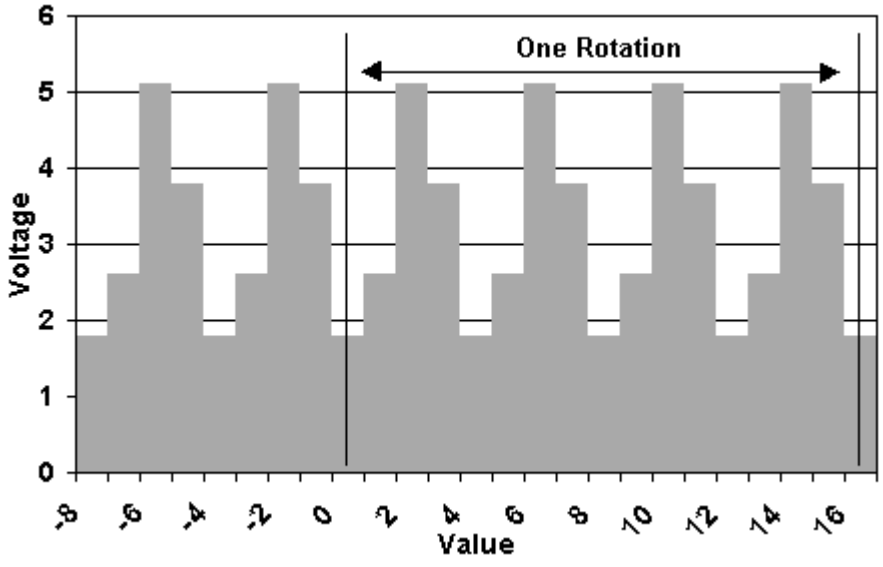


Figura 7.2 Uscite del sensore di rotazione lego

Per la gestione del sensore da parte del legOS ed i comandi ad alto livello si rimanda al paragrafo 5.3.2

Come si è accennato ad inizio capitolo, sarebbe stato naturale sfruttare il sensore messo appunto dalla lego, già costruito e di facile utilizzo ma un'oscura politica della casa costruttrice ne ha bloccato la produzione: in tutta Europa non se ne trova uno[64].
Questa situazione ha reso necessario la realizzazione di un sensore "fatto in casa" (*homemade)*

7.2 Interfaccia di base

[Gasperi 98] Il primo approccio alla costruzione di un sensore consiste nell'individuazione dell'interfaccia che permette l'alimentazione ed il rilevamento dei dati dal sensore, tutto attraverso le porte di ingresso (`1,2,3`)
Come si è visto nel Paragrafo 3.3 Light Sensor ed in particolare in figura 3.3 (sensore aperto), l'alimentazione ed il rilevamento avvengono su uno stesso canale, un sistema a ponte diodi permette poi di diramare massa, alimentazione ed uscita (intesa come uscita del sensore,ovvero segnale di ritorno)
In figura 7.3 è possibile osservare come un segnale analogico venga rilevato dal RCX .
Sono visibili i due cavi che arrivano dal RCX (porte di ingresso), il ponte diodi, il segnale che vogliamo rilevare (Input) collegato attraverso la resistenza R1, e la massa (Gnd)

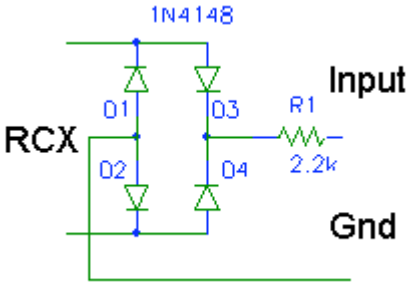


Figura 7.3 Interfaccia di ingresso

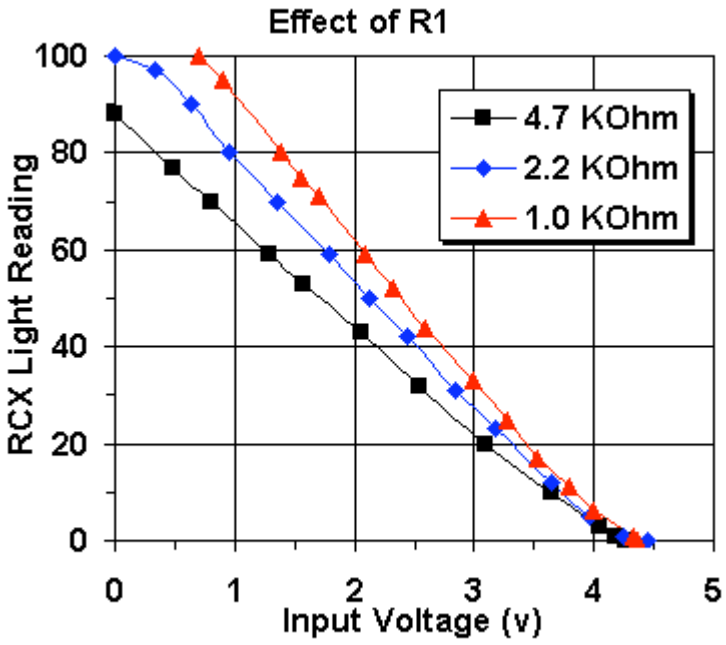


Figura 7.4 Grafico al variare di R1

Nella figura 7.4, viene mostrato l'effettivo valore rilevato, al variare della resistenza , con il sensore inizializzato come Light sensor e range scalato da 0%→5volt a 100%.→0volt
Il valore di resistenza ottimo è 2.2kohm.

[64] Dati resi dall'ufficio centrale di Londra all'autore. Anche i magazzini centrali in Danimarca ne sono sprovvisti.

Se il di segnale di ingresso è minore di 5volt è possibile applicare un amplificatore operazionale:

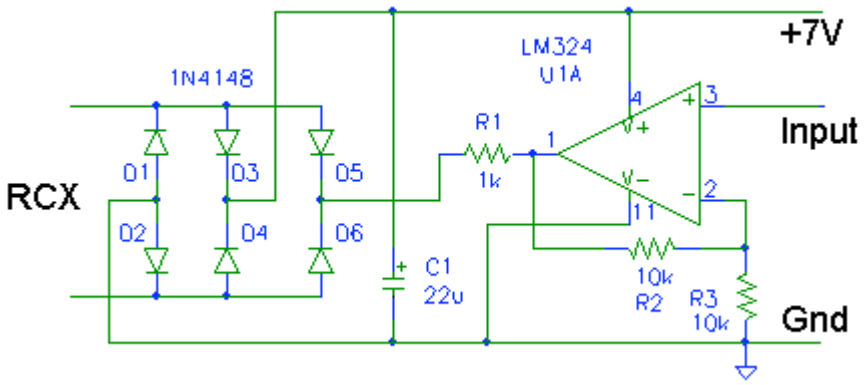


Figura 7.5 Interfaccia di ingresso con Amplificatore Operazionale

Dovendo utilizzare l'Amplificatore sarà necessario alimentarlo, come si nota in Figura 7.5, è stata introdotta una nuova coppia di diodi da cui si preleva l'alimentazione e tra questa e massa si inserisce un condensatore per mantenere il livello di tensione costante (a 7volt) durante il rilevamento, quando l'alimentazione dall'RCX calerà a 5volt.
Il guadagno dell'AO è 1+R2/R3, quindi raddoppia il valore della tensione di ingresso, è possibile anche in questo caso monitorare la tensione rilevata , al variare di R1:

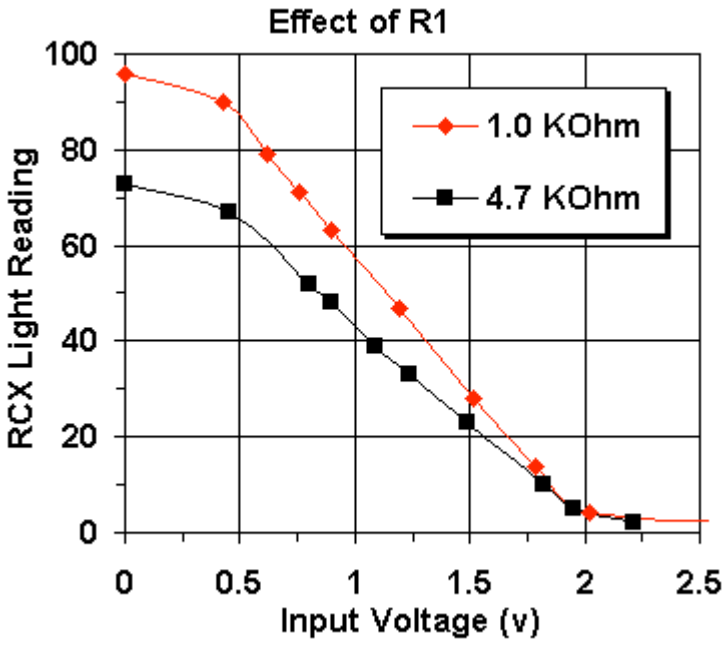


Figura 7.6 Effetto di R1 sul rilevamento delle porte di ingresso

In questo caso si nota una maggiore zone di linearità per R1=1kohm

Effettuando questi esperimenti è possibile concludere che concludere:

- L'interfaccia così costruita viene vista dal RCX come un vero sensore, senza distinzione alcuna; se la porta a cui viene collegata è inizializzata come *light sensor* , l'RCX rileverà i valori come se li leggesse da un sensore di luminosità.
- È possibile utilizzare l'interfaccia anche per dispositivi attivi, che necessitano di alimentazione. Anche per più di un dispositivo, infatti il ramo a 7volt in figura 7.5 provvede a circa 30mA "di potenza"

- È possibile interfacciare il segnale di ingresso con una resistenza.
- All'RCX sono collegati solamente 2 cavi.

7.3 Uso di sola architettura Hardware

Inizialmente si è cercato di costruire un sensore di rotazione complesso che da solo misurasse la velocità di avanzamento del robot, per ottenerlo

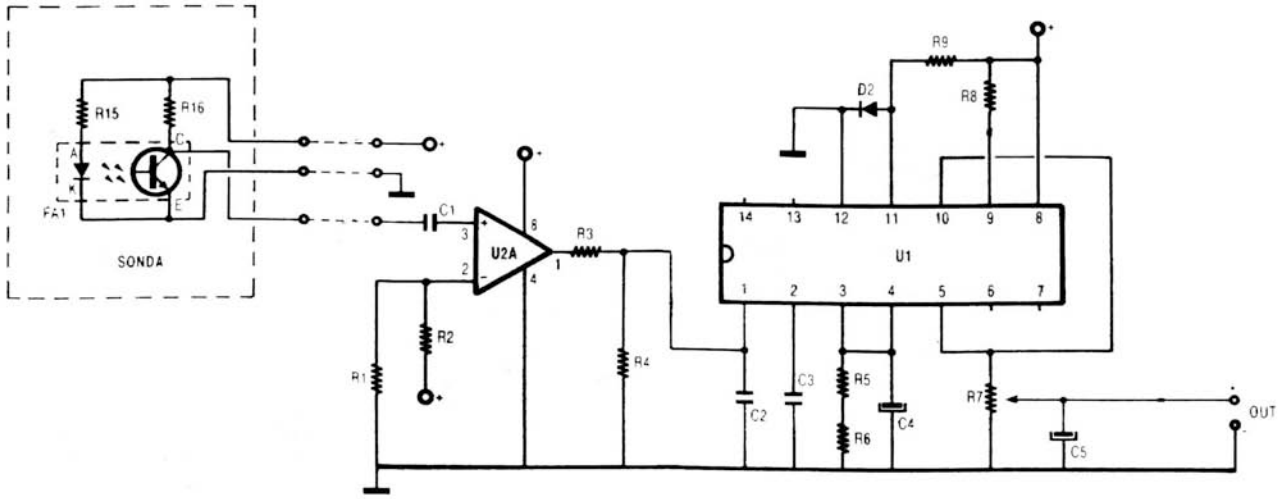


Figura 7.7 Circuito con convertitore F-V

L'alimentazione ed il segnale di uscita vengono prelevati con il circuito analizzato nello scorso paragrafo.

`Sonda`: è il vero e proprio sensore, costituita da un diodo IR (infrarosso) ed da un fototransistor , più due resistenze di polarizzazione.

`C3` :condensatore di accoppiamento

`U2A` :Il segnale prelevato dal collettore viene "ripulito" attraverso un amplificatore operazionale in configurazione da comparatore, essendo R1 e R2 uguali confronta il segnale d'arrivo con Vcc/2. Restituisce quindi un segnale squadrato

`R3,R4`: Sono un partitore di tensione

`C2` :è inversamente proporzionale alla tensione di *ripple,* mentre è diretttamnet proporzionale al tempo che il segnale impiega per stabilizzarsi

`U1` :LM2907N[65] Convertitore frequenza-tensione al cui interno la tensione viene stabilizzata da un diodo Zener a Vz=7.56volt. È costituito da un comparatore (ingresso invertente 11, non invertente 1), da un *charge pump* collegato all'uscita dell'amplificatre ed ai pin 2,3; infine è presente un inseguitore comandato in corrente (ingresso invertente pin 10, non invertente pin 4), alla cui uscita c'è un transistor (BJT, collettore connesso a pin 8) che fornisce 50mA, utili per comandare relè , solenoidi e Led. La retroazione dell'inseguitore è esterna (dal pin 10 al 5)

[65] National semiconductor, LM2907 Frequency to voltage converter

l'uscita (pin 5) restituisce (con linearità del 0.3%)

$$V_0 = V_z \cdot f_{in} \cdot C_3 \cdot (R_5 + R_6) \qquad (7.1)$$

C3 :converte la frequenza con cui varia la tensione ai capi di C2 , in una tensione continua. Quando l'ingresso (Vc_2) cambia di stato il condensatore cresce o decresce linearmente (effetto integratore) tra 2 tensioni che differiscono di Vz/2.

R5,R6,C4 il valore di C4 determina la sensibilità del sensore, mentre le due resistenze influenzano il valore dell'uscita.

R7 :Tara l'uscita ottenute ad un livello che può essere rilevato dagli ingressi dell'RCX

C5 :Stabilizza la tensione di uscita che andrà collegata ai diodi di interfaccia

D2,R9,R8:Servono per settare il valore dell'alimentazione ,della massa ed altre tensioni di riferimento utilizzate dai componenti all'interno dell'integrato

Questo dispositivo viene spesso utilizzato per il controllo di velocità di motori, anemometri etc.. L'idea di impiegare una soluzione quasi completamente hardware si è mostrato utile se si intende effettuare un controllo di velocità del robot, utilizzando anche della retroazione, ma con il robot nella nostra configurazione si è rivelato inefficace.
I motivi possono essere riassunti in

- il robot non compie lunghi percorsi ma come nel caso del *follow-wall* assesta continuamente la sua posizione, avanzando ed arretrando, in alcune occasioni senza compiere nemmeno un giro
- L'uso del sensore deve essere versatile per poter essere impiegato anche con il *grasp,* in modo da sapere quando le ´braccia´ del robot hanno spaziato l'angolo necessario per chiudersi le su un ogetto. Ovvero deve poter essere utilizzato non solo some sensore per l'odometria ma anche come vero e proprio sensore di rotazione angolare
- La scheda è ingombrante, la sua lunghezza è pari a quella dell'RCX.

Per queste ragioni si è ritenuto opportuno implementare una sensoristica più elementare, controllabile maggiormente via software.

7.4 Architettura hardware

Come anticipato alla conclusione dello sorso paragrafo, si è puntato ad un sensore dall'hardware molto semplice e dal controllo software molto pesante.
Del circuito di Figura 7.7 si è tenuta la sonda ,e l'interfaccia di collegamento alle porte di ingresso dell'RCX vista nel paragrafo 7.2

7.4.1 Schema elettrico

È possibile notare lo schema del sensore di rotazione in Figura 7.8, costituito da:

- `6 diodi Schottky:` per l'interfaccia di base si sono utilizzati diodi a barriera Schottky per la loro caratteristica di avere tensione di soglia V_γ molto bassa ed corrente inversa di saturazione più alta; queste caratteristiche gli permettono di compire commutazioni in tempi più rapidi di un diodo a giunzione .

  Infatti trattandosi di un dispositivo la cui giunzione è formata da un metallo e da un semiconduttore, il funzionamento è basato sui portatori di carica maggioritari (in un metallo non ci sono portatori minoritari), quindi il tempo di immagazzinamento è trascurabile mentre il tempo di ripristino inverso include soltanto il tempo di transizione[66].

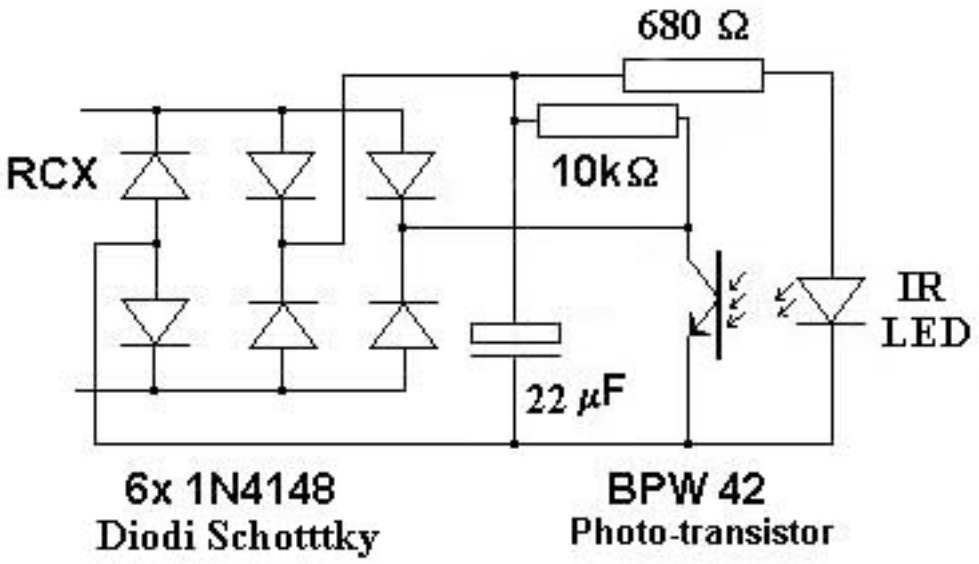


Figura 7.8 Rotation Sensor

- `Condensatore 22uF:` Il condensatore viene utilizzato per stabilizzare la tensione ai capi di rami che costituiranno l'alimentazione della sonda, ovvero l'alimentazione al transistor ed al diodo attraverso le resistenze.

- `IR LED:` Il LED infrarosso, è un diodo che emette radiazioni nel campo infrarosso, è connesso tra massa e l'alimentazione attraverso una resistenza da 680 ohm

- `Photo Transistor:` Il transistor è sensibile esclusivamente all'infrarosso, ha una resistenza di circa 100kohm quando è attivo, ed 1Mohm quando non è attivo. Fisicamente ha la forma di un transistor , con 3 fili per i collegamenti di cui uno non viene usato, quello della Base, e nella sommità è bombato e trasparente per permettere il rilevamento. Vi è un altro tipo di photo transistor, che non è intercambiabile con quello usato nel sensore, esteriormente si distingue dall'avere soltanto 2 fili , mentre fisicamente la distinzione consiste nel poter rilevare sia il campo infrarosso sia il campo della luce visibile infine la sua resistenza è circa 100 volte minore.

7.4.2 Funzionamento

Il transistor viene attivato dalla luce infrarossa emessa dal Led IR. Questa attivazione non è di tipo On/Off ma produce una passaggio di corrente nel transistor tanto maggior, tanto minore è la distanza tra i due componenti;

[66] Per ulteriori dettagli vedasi [Millman, Grabel 97]

NOTA: Il transistor si attiva con qualsiasi segnale infrarosso,quindi anche con quelli emessi dai dispositivi presenti in casa: telecomandi etc….È opportuno che il sensore sia disposto in modo tale da non permettere che ciò accada

Il segnale rilevato viene preso dal collettore del transistor e instradato all'RCX attraverso i diodi di interfaccia. La misura della tensione misurata dipenderà dalla corrente che circola nel transistor:

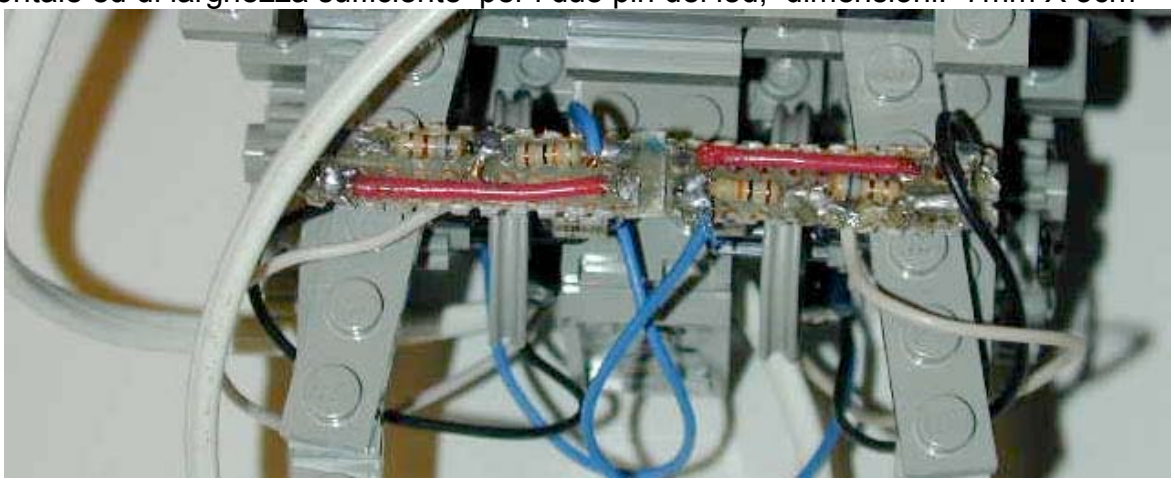
$$V_{rilevata} = V_{cc} - i \cdot R_1 \qquad (7.2)$$

dove:
Vcc :rappresenta la tensione di alimentazione,
i :rappresenta la corrente
R1 rappresenta la resistenza di Pull-up del transistor, 10kohm
Si osserva che, se il transistor andasse in saturazione la tensione rilevata sarebbe la Vce di saturazione ovvero 0.2volt

La seconda fase di funzionamento avviene quando qualche oggetto viene posto tra il diodo ed il transistor, in questo caso il transistor non sarà attivo ed il ramo in cui è connesso equivarrà ad un ramo aperto. La tensione rilevata grazi alla resistenza di pull-up sarà Vcc = 5volt.
Queste due fasi del photo transistor, attivo e non attivo, vengono sfruttate per rendere il sensore di rotazione un encoder incrementale, ovvero un contatore di eventi, e gli eventi consistono in rilevare valori pari a Vcc ed oppure ad una tensione più bassa.

Realizzazione circuitale

Per montare il circuito non si utilizza un'unica basetta, la parte della sonda viene montata separatamente ed collegata all'altra per mezzo di 3 fili. La parte della sonda comprende le 2 resitenze , il fototransistor ed il led IR.
Essendo necessari 2 sensori di rotazione , sono necessari 2 circuiti. Le 2 interfacce per l'RCX sono state montate su un'unica basetta larga 3cm e alta 2cm, a cui vengono collegate le due sonde, ed i cavi dalle porte dell'RCX .
Ogni sonda è montata su una basetta di lunghezza tale da contenere le 2 resistenze in orizzontale ed di larghezza sufficiente per i due pin del led, dimensioni: 7mm X 3cm



7.5 Architettura meccanica

La forma fisica del led e del photo transistor si presta molto bene ad essere usata con i lego, e più specificatamene con *pulley* (pulegge) di larghezza media[67], le *pulley* assomigliano ad un disco forato, con 6 fori (hole) della larghezza poco più piccola di un led. Come mostrato in Figura 7.9 vengono fruttati questi fori per far passare o meno la luce infrarossa prodotta dal Led, la configurazione ideale è led e transistor perfettamente allineati e separati soltanto dalla presenza della *pulley* che non li urta.

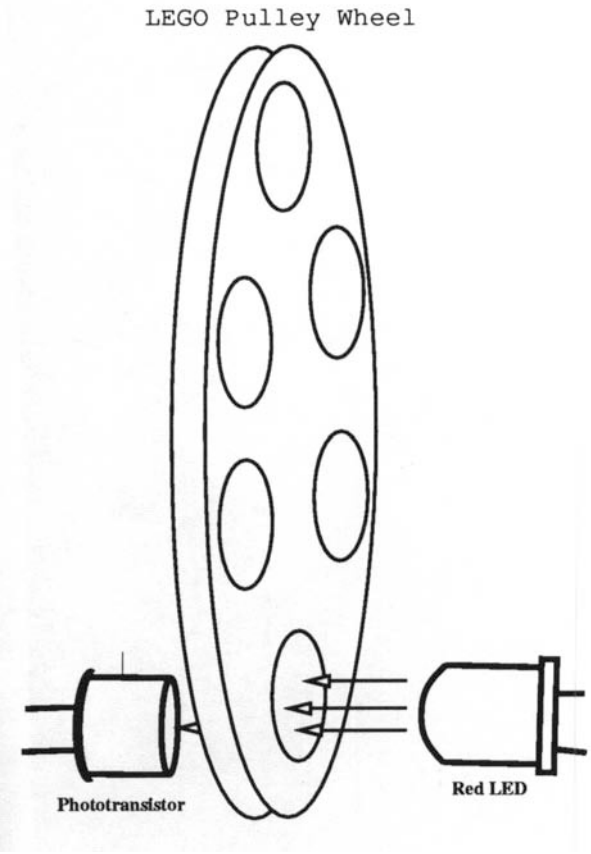


Figura 7.9 Utilizzo dei fori dei pulley

Quanto mostrato in questo schizzo risulta evidente nella Figura 7.10 in cui è possibile notare, dall'alto verso il basso,

- prima le basette che contengono le 2 resistenze da un lato ed il transistor (colore metallico) e il Led IR (colore blu) dall'altro.
- Il mezzo ai due componeti si vede la pulley,
- All'asse delle pulley sono connessi una ruota dentata da 8, e subito sotto una da 16.
- Nel fondo della figura si nota il circuito stampato, contenente le due interfacce per il collegamento agli ingressi dell'RCX, è tenuto saldamente ancorato con un blocco lego fatto a "L". Si scorgono anche due cilindri blu, posti in modo simmetrico, sono i condensatori.
- I cavi banchi che si intravedono portano ai connettori che si collegano alle porte di ingresso

[67] si rimanda a paragrafo 2.1.2

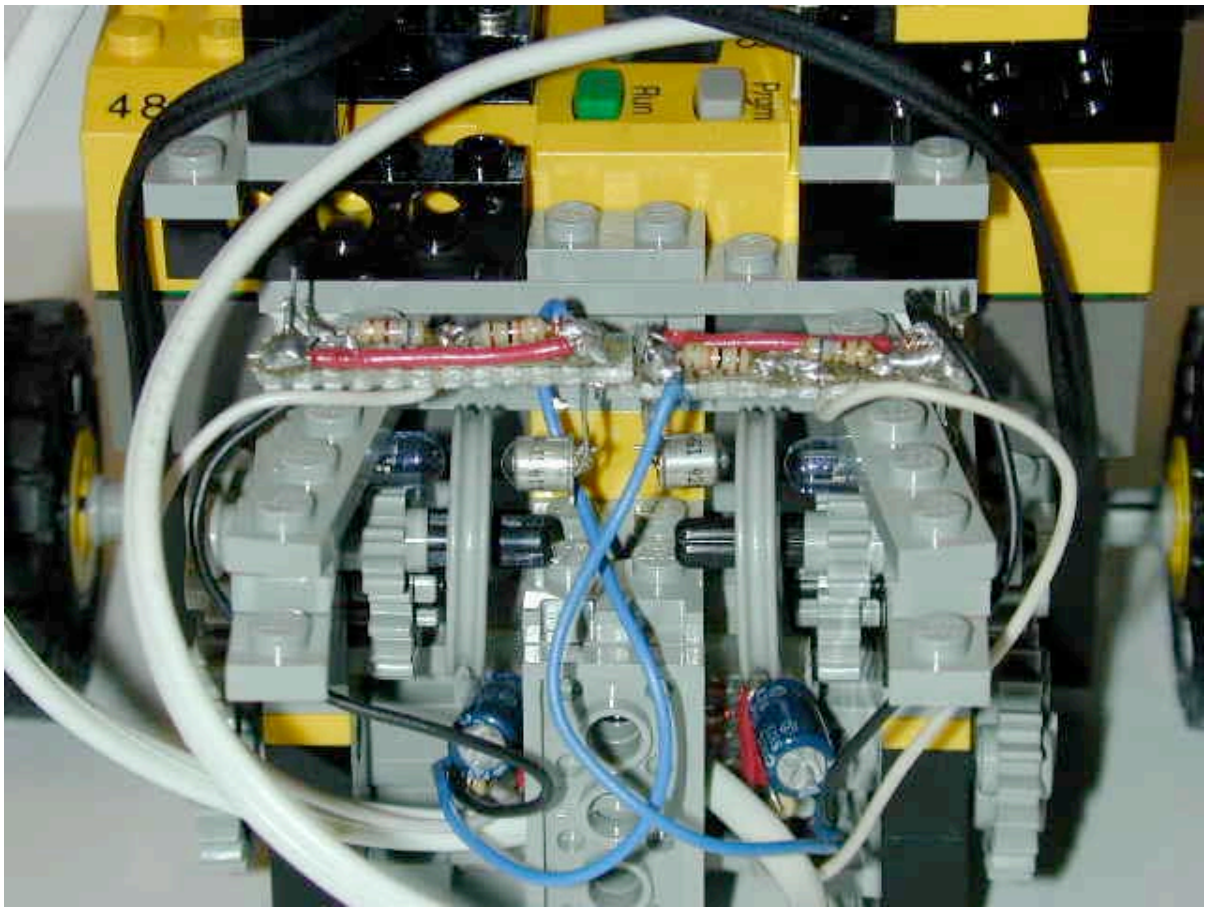


Figura 7.10 Sensore di rotazione

Per ottenere un sensore di rotazione con la sensibilità di 10° è necessario un rapporto di trasmissione tale che per ogni giro delle ruote motrici , davanti al sensore passino 36 *hole*; essendo ogni *pulley* con 6 fori, equivale ad un *gear up* pari a 1:6.
In Figura 7.11è possibile notare come tutto questo sia realizzabile attraverso un *gear train*, collegando al lato "libero" dell'"ingranaggio motore" da 24 denti,(dall'altro lato vi sono le ruote motrici), una ruota dentata da 24 denti, connessa ad una da 8 , sul cui asse è inserita una da 16 a cui è collegata un'altra da 8 sul cui asse viene inserita la *pulley:*
1:1 x 1:3 x 1:2 = 1:6 Per ogni giro della ruota motrice l'asse delle *pulley* compie 6 giri.

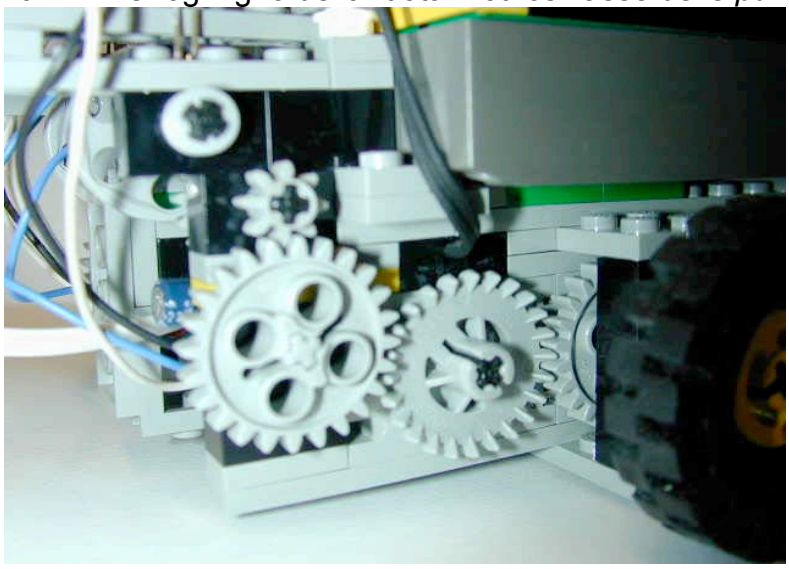


Figura 7.11 gear train

NOTA:Non basta conoscere che ruote dentate utilizzare è indispensabile collegarle in modo da rendere la posizione del sensore in zone in cui non può essere danneggiato e dal moto oda urti esterno.; in figura si vede come la posizione del sensore e degli ingranaggi è tale da essere fuori dalla sagoma del robot, in modo da evitare urti laterali, mentre il moto verso dietro non viene mai utilizzato dai programmi.

Sono stati costruiti in casa anche i connettori per le porte di input dell'RCX , per visionarli si rimanda alla figura 9.5 al Capitolo 9.2.2

7.6 Implementazione software

Nell'implementare il counter incrementale si preferisce non contare le *hole* ma le zone plastiche tra un foro e l'altro perché è su queste che viene letto il valore 5volt.
Il sensore della ruota destra viene collegato al canale 3 mentre quello sulla ruota sinistra al canale 1; le porte di ingresso vengono settate per leggere il valore quantizzato (`0-1023, 1023=5volt);` quando passa una *hole* davanti al IR led viene rilevato un valore che varia dai 3 ai 4 volt, quindi più di un volt di margine.

Nel seguito verranno illustrate alcune implementazione che risolvono in successione gli errori incontrati fino all'ultima che non ne presenta.

7.6.1 Contatore come task a singola soglia

L'idea iniziale è utilizzare due task indipendenti , per il sensore destro e sinistro, che contino ogni qual volta nella porta di ingresso associata compare un valore di 5volt.
I due task si chiamano
`count_r_hole()`
`count_l_hole()`
Il conteggio avviene utilizzando un automa a stati finiti a 2 stati che valgono "1" per indicare la presenza dei 5volt e "0" per indicare un qualsiasi altro voltaggio .

Nel seguito viene riportato un estratto del codice, per un solo task, l'altro è simmetrico.
Le soglie per le transizioni distato vengono fissate a:
`MAX_L`, soglia sinistra
`MAX_R`, soglia destra, entrambe a 5000 millivolt
Mentre le soglie inferiori sono il valore massimo meno un certo Epsilon (eps) fissato a 100
La funzione `mv1()` ritorna il valore in volt del segnale rilevato nella porta 1.
Il valore di conteggio si chiama `l_h`

```
Dal file ODO2.C
\author Maragno Simone simomail@inwind.it

      #define MAX_L  0x1388   // 5000 mvolt
      #define MAX_R  0x1388
      #define eps        0x64             //    100
      #define t_l        (int) MAX_L-eps  //threshold ruota  sinistra
      #define t_r        (int) MAX_R-eps

void   count_l_hole(){
  int state=0,mv,i=0;                              //l_h: #left hole,
  state=(mv1()>t_l)?1:0;
  while(1){
      mv=mv1();
```

```
      if((s>0)&&(t==1)) lcd_int(l_h);
      switch (state) {
            case 0:
            if (mv>(t_l)){
                  l_h+=dir_l;
                  //sem_getvalue(&semlh,&lh);
                  //sem_post(&semlh);
                  state=1;
            }break;
        case 1:
                  if (mv<=(t_l))  state=0;
            }
    }
   }
```

Osservazioni:

- I valori di `l_h, r_h` differiscono di molto
- Il calcolo del valore in volt porta via tempo

7.6.2 Contatore come task a doppia soglia

La soluzione trovata è usare un sistema tipo trigger a due soglie :
`LEFT_THRESHOLD :` 4500mv
`RIGHT_THRESHOLD :` 4900mv

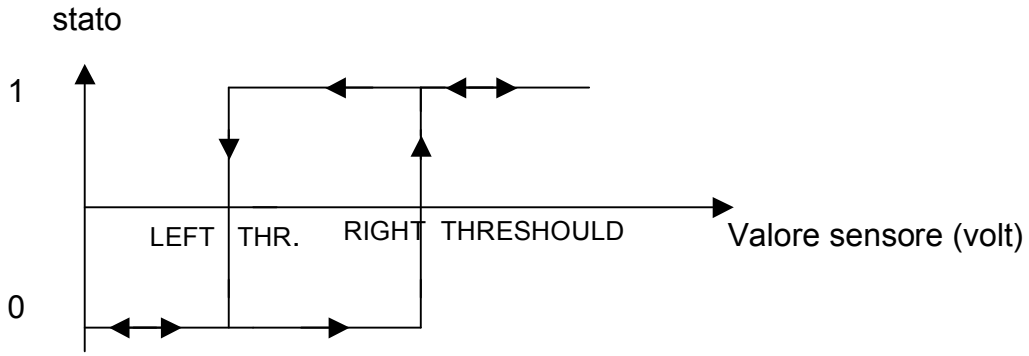


Figura 7.12 sistema a 2 soglie

Il conteggio avviene nel passaggio “0-1”, quando si superano per la prima volta i 4.9 volt

Di seguito un estratto del codice, come nello scorso paragrafo viene riportato solamente un task

```
count_l_hole(){
  int state=0,i=0,mv,cont=0;                                  //l_h: #left hole,
  while(1)  {
  //mv=mv1;
  //  if((s>0)&&(t==1)) lcd_int(l_h);
  if (mv1 <= LEFT_THRESHOLD)
            {
                  while (mv1 < LEFT_THRESHOLD) ;
            }
```

```
            else if (mv1 >= RIGHT_THRESHOLD)
            {
                  while(mv1 > RIGHT_THRESHOLD);
                  l_h+=dir_l;
            }
   }
}
```

Osservazioni:

- I valori indicati da `l_h e r_h`, sono uguali
- Una verifica più accurata mostra che a velocità non lente, le ruote girano molto piú veloce del valore contato:
  A 3.7 giri /sec (speed=50) 36*3.7=133 conteggi, quelli rilevati sono poco superiori a metà
- Lo scorso esempio rileva a necessità di produrre un campione ogni 7.5ms. Ma ogni task rimane attivo solamente per alcuni millisecondi.

7.6.3 Modifica kernel: rotation_handler

Tempi di campionamento così bassi portano alla necessità di lasciare la struttura task, e lavorare ancora più a basso livello, operando sugli interrupt_handler[68], ovvero a modificare i file

`/include/dmotor.h`
`/include/dsensor.h`
`/kernel/dsensor.c`

`ds_rotation_set:`
È una funzione modificata in `/dsensor.c` , inizializza il sensore, prendendo i ingresso:

- La porta a cui è collegato il sensore
- Il tipo di utilizzo che si intende effettuare, per l'odometria oppure per il controllo di braccia meccaniche
- l'angolo di partenza da cui inizia la rotazione (nell'odometria è zero)
- la soglia destra e sinistra: LEFT_THRESHOLD e RIGHT_THRESHOLD

Le nuove variabili introdotte sono tutti array da 3 elementi, (uno per ogni porta di input), particolarmente degne di nota sono `ds_rotation_dir` e `dir2count`

- `ds_rotation`:tiene il conteggio
- `rotation_left_threshold`: indica il valore di soglia sinistra, non in volt, è un valore esadecimale
- `rotation_right_threshold`: valore di soglia destra
- `rotation_use`: indica come vogliamo utilizzare il sensore
- `Rotation_new_state`: indica il nuovo stato in cui si è portato il sensore, (utilizziamo sempre un contatore a soglie)
- `ds_rotation_dir`: indica la direzione di rotazione delle ruote , non essendoci un metodo come nel sensore lego per rilevare il verso di rotazione, è necessario ottenerlo direttamente quando i motori cambiamo verso, non attraverso un controllo software, ma attuando la modifica direttamente nel "comando dei motori"
  Questa variabile viene direttamente importata dal file `/dmotor.h` , ed assume un valore da 0 a 3 a seconda della direzione : `off, fwd, rev, brake`

[68] Vedasi paragrafo 5.3.2 Sensor control

- `dir2count`: Si occupa di convertire il valore della variabile `ds_rotation_dir`, in un fattore moltiplicatore che permetterà di incrementare, diminuire o lasciare invariato il conteggio a seconda che la direzione sia avanti, indietro o Off/brake:
  Off → 0
  Fwd → +1
  Rev → -1
  Bake → 0

Di seguito un estratto del file `dmotor.h` in cui si nota l'aggiunta della variabile `ds_rotation_dir[]` all'interno della funzione `motor_a_dir(dir)`.
Si ricorda che vi è una funzione siffatta per ogni uscita dei motori (A,B,C), in questo estratto è riportata solamente quella per il motore A.
NOTA: si assume che il:

- sensore di rotazione 1 collegato alla porta 1 misura il motore collegato alla porta A
- sensore di rotazione 2 collegato alla porta 2 misura il motore collegato alla porta B
- sensore di rotazione 3 collegato alla porta 3 misura il motore collegato alla porta C

```
/*! \file   include/dmotor.h
    \brief  direct motor access
    \author Markus L. Noga <markus@noga.de>
            Maragno Simone <simomail@inwind.it>
 *
 *          included ds_rotation_dir[] into motor_a_dir( dir)
 */

//! the motor directions
typedef enum {
  off = 0,              //!< freewheel
  fwd = 1,              //!< forward
  rev = 2,              //!< reverse
  brake = 3             //!< hold current position

//! rotation variables
extern volatile char ds_rotation_dir[3];

//! set motor A direction
/*! \param dir the direction
 */
/*
 * Modify the line if the ROTATION_X isn't connected to MOTOR_X 's wheel
 *    Note:             ds_rotation_dir[2] rotation sensor in SENSOR_1
 *                      ds_rotation_dir[1] rotation sensor in SENSOR_2
 *                      ds_rotation_dir[0] rotation sensor in SENSOR_3
 */
extern const inline void motor_a_dir(MotorDirection dir)
{
  ds_rotation_dir[2]=dir;          // if Rotation is in SENSOR_1 :
                                   // ROTATION_1 connected to MOTOR_A 's wheel
  dm_a.dir = dm_a_pattern[dir];
}
```

Considerando il file `/dsensor.c`, la funzione `ds_sensor()`, scritta in assembler chiama il salto alla gestione dell'interrupt del sensore di rotazione:

```
jsr _ds_rotation_handler ; process rotation sensor
```

quindi viene attivata la funzione `ds_rotation_handler();` è stato necessario riscriverla tutta, in quanto il nostro sensore non si comporta come quello della lego.

La funzione una volta inizializzato il canale ,il convertitore A/D necessario ad eseguire la conversione e settato il valore letto come valore quantizzato (`0-1023`), si procede al conteggio delle zone piene (non delle *hole*); il metodo di conteggio è sempre a trigger.

Nel seguito viene riportato un estratto da file `dsensor.c`, in cui si possono notare uttte le variabili aggiunte e le due funzioni `ds_rotation_handler()` e `ds_rotation_set`, alla fine è presente anche una versione di contatore a singola soglia, poi abbandonato perché eseguiva un sovraconteggio.

```
/*! \file   dsensor.c
    \brief  direct sensor access
    \author Markus L. Noga markus@noga.de

 *     - Maragno Simone <simomail@inwind.it>
 *
 *    - modify/rewrite ds_rotation_set, ds_rotation_handler  for
 *          homemade rotation sensor
 *    - new variable:         rotation_left_threshold, rotation_right_threshold
 *                            rotation_use, ds_rotation_dir
 */

#include <dsensor.h>

#ifdef CONF_DSENSOR

#include <rom/registers.h>
#include <sys/h8.h>
#include <sys/irq.h>
#include <sys/bitops.h>
//#include <rom/registers.h>
#include <unistd.h>
#include <conio.h>

////////////////////////////////////////////////////////////////////////////
//
// Definitions
//
////////////////////////////////////////////////////////////////////////////

#define DS_ALL_ACTIVE         0x07             //!< all sensors active mode
#define DS_ALL_PASSIVE        (~DS_ALL_ACTIVE) //!< all sensors passive mode

////////////////////////////////////////////////////////////////////////////
//
// Variables
//
////////////////////////////////////////////////////////////////////////////

volatile unsigned char ds_channel;        //!< current A/D channel
unsigned char ds_activation;              //!< channel bitmask. 1-> active

#ifdef CONF_DSENSOR_ROTATION
unsigned char ds_rotation;                //!< channel bitmask. 1-> rotation
volatile int ds_rotations[3];             //!< sensor revolutions in deg
```

```
volatile char ds_rotation_dir[3];  // direction wheel-->written into motor.h
static int rotation_left_threshold[3];         // threshold
static int rotation_right_threshold[3];
static unsigned char rotation_use[3];     //!< rotation use change
static unsigned  rotation_new_state[3]; //!< proposed rotation statech.



//! convert direction motor to increment count hole
/*!
*/
static const signed char dir2count[4]={
  // 0  1  2  3                        // dir motor: off, fwd, rev, brake
     0, 1, -1, 0
};

////////////////////////////////////////////////////////////////////////////
//
// Functions
//
////////////////////////////////////////////////////////////////////////////

void ds_rotation_set(volatile unsigned *sensor,RotationUse use, int pos,int
LEFT_THRESHOLD,int RIGHT_THRESHOLD)
{
  if(sensor>=&AD_A && sensor<=&AD_C) {    // catch range violations
    unsigned channel=(unsigned) (sensor-&AD_A);
    int  state=(*sensor)>>6;                          //// 0-1023

            rotation_left_threshold[channel]=   LEFT_THRESHOLD;
            rotation_right_threshold[channel]=RIGHT_THRESHOLD;
    rotation_use[channel]=use;
    rotation_new_state[channel]=0;
    ds_rotations[channel]=pos;            // set to pos

  }
}

//! process rotation sensor on current A/D channel
/*! \sa ds_channel current channel (global input value)
*/
void ds_rotation_handler() {
  unsigned    channel =ds_channel;
  unsigned    raw     =(*((&AD_A)+channel));
  unsigned  val=(raw>>6);
  signed char   dir=dir2count[ds_rotation_dir[channel]];              //

  if (rotation_new_state[channel]==1) {
                            //a trigger
      if(val<=rotation_left_threshold[channel])
            rotation_new_state[channel]=0;
      }
  else if (val>rotation_right_threshold[channel]){
                ds_rotations[channel]+=dir;
                rotation_new_state[channel]=1;
            }
 }
//
//versione con una sola soglia: conta  piú giri.
//
  /*if (rotation_new_state[channel]==1) {
```

```
      if (val<=rotation_right_threshold[channel])
            rotation_new_state[channel]=0;
            }
      else if (val>rotation_right_threshold[channel]){
                  ds_rotations[channel]+=dir;
                  rotation_new_state[channel]=1;
            }
 }*/

#endif // CONF_DSENSOR_ROTATION
```

7.6.4 Interfaccia ad alto livello

A livello del programmatore, sono stati creati due nuovi comandi per gestire il sensore di rotazione, visibili nel include file `/dsensor.h`

`rotation_x_set( use, position)`
È una macro che permette l'inizializzazione del sensore di rotazione, in ingresso prende il modo in cui si intende usare il sensore (odometria o misuratore di angoli) , e la posizione angolare iniziale. Dove x sta per il numero della porta di input

`rotation_x_on(&SENSOR_x )`
Attiva il sensore di rotazione, da lanciare ad inizio programma, in pratica fa partire l'alimentazione al sensore. Nell'edizione lego la cosa non era automatica, una volta inizializzato il sensore era necessario lanciare `ds_active(&SENSOR_x)` per attivare la porta a cui era connesso il sensore.

`ROTATION_X`
È la macro con cui viene gestito il valore del sensore di rotazione, (sensore collegato alla porta x), contenedo un intero può essere utilizzata anche come argomento per altre funzioni come ad esempio per stampare il valore sull'lcd
`lcd_int(ROTATION_2)`

Di seguito un estratto del file `/dsensor.h`, il programmatore può anche modificare i valori delle soglie qui riportati
`RIGHT_ROT_x_THRESHOLD`
`LEFT_ROT_x_THRESHOLD`

```
/*! \file   include/dsensor.h
    \brief  direct sensor access
    \author Markus L. Noga <markus@noga.de>
            Maragno Simone <simoimail@inwind.it>
 *
 *    - Some modify for homemade rotation sensors, view "kernel/dsensor.c"
 *          LEFT_ROT_THRESHOLD, RIGHT_ROT_THRESHOLD, typedef RotationUse,
 *
 *    -defined new user interface macro functions:
 *     rotation_x_set(use, position)
 *     rotation_x_on    for activate rotation sensor and power to the sensor
 */

#ifdef CONF_DSENSOR_ROTATION
```

```
//
// processed rotation sensor
//
#define LEFT_ROT_1_THRESHOLD   0x399    //(921 quanti   4500 mv)
#define RIGHT_ROT_1_THRESHOLD  0x3eb    //(1003 quanti  4900 mv)
#define LEFT_ROT_3_THRESHOLD   0x399    //( quanti   4500 mv)
#define RIGHT_ROT_3_THRESHOLD  0x3eb    //(  quanti  4900 mv)

#define ROTATION_1  (ds_rotations[2])
#define ROTATION_2  (ds_rotations[1])
#define ROTATION_3  (ds_rotations[0])

//
typedef enum {
      odometry = 1,
      angle =0,
}RotationUse;

// /param use: what can i do with sensor? odometry or angle mesurement
//                      pos: initial position
#define rotation_1_set(use,pos) (void)ds_rotation_set( &SENSOR_1,use, pos,
LEFT_ROT_1_THRESHOLD, RIGHT_ROT_1_THRESHOLD)
#define rotation_2_set(use,pos) (void)ds_rotation_set( &SENSOR_2,use, pos,
LEFT_ROT_1_THRESHOLD, RIGHT_ROT_1_THRESHOLD)
#define rotation_3_set(use,pos) (void)ds_rotation_set( &SENSOR_3,use, pos,
LEFT_ROT_3_THRESHOLD, RIGHT_ROT_3_THRESHOLD)

// activate rotation sensor interrupt handler and power at sensor_x
#define rotation_1_on
(void)ds_rotation_on(&SENSOR_1);(void)ds_active(&SENSOR_1)
#define rotation_2_on
(void)ds_rotation_on(&SENSOR_2);(void)ds_active(&SENSOR_2)
#define rotation_3_on
(void)ds_rotation_on(&SENSOR_3);(void)ds_active(&SENSOR_3)

#endif

#ifdef CONF_DSENSOR_ROTATION

extern unsigned char ds_rotation;       //!< rotation   bitmask

extern volatile int ds_rotations[3];     //!< rotational position

#endif

extern void ds_rotation_set(volatile unsigned *sensor,RotationUse use,int
pos,int LEFT_THRESHOLD,int RIGHT_THRESHOLD);

//! start tracking rotation sensor
/*! \param  sensor: &SENSOR_1,&SENSOR_2,&SENSOR_3
*/
extern inline void ds_rotation_on(volatile unsigned *sensor)
{
  if (sensor == &SENSOR_3)
    bit_set(&ds_rotation, 0);
  else if (sensor == &SENSOR_2)
    bit_set(&ds_rotation, 1);
  else if (sensor == &SENSOR_1)
    bit_set(&ds_rotation, 2);
}

//! stop tracking rotation sensor
```

```
/*! \param  sensor: &SENSOR_1,&SENSOR_2,&SENSOR_3
*/
extern inline void ds_rotation_off(volatile unsigned *sensor)
{
  if (sensor == &SENSOR_3)
    bit_clear(&ds_rotation, 0);
  else if (sensor == &SENSOR_2)
    bit_clear(&ds_rotation, 1);
  else if (sensor == &SENSOR_1)
    bit_clear(&ds_rotation, 2);
}

#endif // CONF_DSENSOR_ROTATION
```

Esempi di programmazione possono essere trovati in Appendice.

Capitolo 8 Funzioni matematiche

Funzioni matematiche

In questo capitolo vengono illustrati i metodi implementati per poter utilizzare il sensore di posizione ovvero per poter elaborare i rilevamenti da esso compiuti.
Il microcontrollore della Hitachi , cuore del RCX , non possiede coprocessore matematico quindi la trigonometria necessaria non viene fornita via Hardware, e nemmeno via software in quanto il legOS , oltre alle semplici operazioni (+,*,-,/,>,<, &,|) ed un generatore di numeri random dispone solamente di un emulatore di numeri float, mentre per gli interi vengono utilizzati 16 bit.
Per sfruttare completamente i 16 bit viene utilizzata la matematica in notazione Fixed Point (a virgola fissa). Per implementare la funzione *sin(x)* sono stati analizzati 3 metodi utilizzando la serie di Tylor, la trasf Zeta e la memorizzazione su un array.

8.1 Binary Fixed Point

[Linkemann 96, Mian 99] L'aritmetica in virgola fissa viene spesso utilizzata nei DSP e nei Mirocontrollori essendo più veloce da gestire, non ci sono errori nelle operazioni di somma se il risultato appartiene ad un numero rappresentabile, se ciò non avviene può accadere un overflow; mentre nelle operazioni di moltiplicazione si compie un errore di arrotondamento/troncamento.

Quanto: nel processo di quantizzazione durante la conversione A/D dicesi il più piccolo numero rappresentabile

$$q = \frac{2M}{2^{n+1}}$$

dove M è il valore del fondo scala, ovvero si considerano valori compresi tra [-M , +M], ed *n* è il numero di bit con cui si rappresenta ogni cifra (*+1* per il segno)
Questo è utile nel caso vogliamo elaborare i dati e poi restituirli mediante un D/A di pari fondo scala.
Nel nostro caso gli ingressi sono quantizzati a 10 bit ed il fondo scala è [0 5volt], ma l'elaborazione che si opera non lavora con i valori in tensione. Infatti come si è visto nello scorso capitolo il sensore di rotazione conta il numero di *hole*, da queste si può dedurre l'angolo spaziato dalla ruota con una sensibilità di 1° .
Il valore di cui disponiamo, quindi, è un numero intero ed è scorrelato dal fondo scala del A/D.
Assumeremo come quanto il più piccolo numero rappresentabile con una notazione a 16 bit (per gli `unsigned`) oppure 15 bit (+1 di segno per i `signned`)

Notazione in virgola fissa S.X.Y

La notazione in virgola fissa si può rappresentare in questo modo dove

- **S** indica la presenza di un bit di segno
- **X** indica il numero di bit che vengono utilizzati per la parte intera
- **Y** indica il numero di bit che vengono utilizzati per la parte frazionaria

È sottointeso che "*X*+*Y*= *n*" , con *n* numero di bit per rappresentare il numero +1 di segno , (se c'è). Il valore in decimale sarà espresso da:

$$dec = bin_{10} \cdot 2^{-y}$$

Con tale scrittura si intende il numero binario, convertito in decimale e moltiplicato per l'opportuna potenza di 2
Per trasformare invece un numero da decimale a binario basterà verificare preliminarmente che la parte intera sia minore di 2^x, moltiplicarlo per 2^y, infine convertire il risultato un binario.

Accuratezza: l'accuratezza di questa conversione è pari all'errore di arrotondamento della parte frazionaria : q/2, con *q* calcolato per un fondo scala pari ad 1 ed *n* (numero di bit) pari a *y:*

$$\frac{q}{2} = \frac{1}{2^y} \cdot \frac{1}{2}$$

per esempio:
Supponiamo di utilizzare 16 bit nella notazione S.7.8 quindi 1bit di segno, 7 per la parte intera ed 8 per la decimale. La risoluzione dipende dall' LSB (bit meno significativo), il più piccolo valore rappresentabile

$$\frac{q}{2} = \frac{1}{2^y} \cdot \frac{1}{2} = \frac{1}{2^8} \cdot \frac{1}{2} = \frac{1}{256} \cdot \frac{1}{2} = \frac{0,00390625}{2} = 0,001953125 \qquad (8.1)$$

quindi l'errore di arrotondamento, ovvero l'accuratezza è di circa 2^{-3}.
Si continua con alcuni esempi di conversione e di somma

$$12,1_{10} \rightarrow 12,1 \times 256 = 3097,6 \rightarrow round \rightarrow 3098 \rightarrow_{16} 0 \times 0C1A$$

$$5,71_{10} \rightarrow 5,71 \times 256 = 1461.76 \rightarrow round \rightarrow 1462 \rightarrow_{16} 0 \times 05B6 \qquad (8.2)$$

$$0 \times 0C1A + 0 \times 05B6 = 0 \times 110D \rightarrow 4560 \rightarrow \frac{4596}{256} = 17,8125_{10}$$

Una volta effettuata la somma si nota come l'errore sia dell'ordine dei millesimi, come stimato: 17,8125 al posto di 17,81, errore di 0.0025.

Moltiplicazioni: I vantaggi di questo modo di interpretare i numeri in binario svengono alla luce quando si devono effettuare operazioni di moltiplicazione o divisione; nelle moltiplicazione si ricorda che due fattori da 16 bit ne generano uno da 32 bit, per tornare a 16 bit basterà prendere solamente i MSB (i 16 bit più significativi) della long Word ottenuta, attraverso uno *shift* matematico.[69]
NOTA: È indispensabile che la moltiplicazione della parte intera del numero non abbia prodotto overflow, ovvero se prima era rappresentabile a *x* lo deve essere anche adesso, altrimenti il metodo dello *shift* non è più realizzabile.

Un notazione molto usata nella programmazione dei DSP, in cui la regola dello *shift* è sempre applicabile è la **s.0.15** indicata più brevemente come **0.15:**

[69] In poche parole memorizziamo gli MSB nel posto degli LSB

- 15 bit per la rappresentazione
- 1 bit di segno
- fondo scala 1

in tale maniera si moltiplicano sempre frazioni dell'unità che come risultato restituiscono sempre un'altra frazione dell'unità
NOTA:se i due numeri sono con il segno basterà effettuare uno *shift* di 15 bit e non di 16bit, guadagnando 1 bit :
Indicando in modo compatto s15 i 16 bit (15 +1 disegno), eseguendo una moltiplicazione si otterrà:
s15 x s15 = ss30 → 32 bit di cui 2 di segno , ma a noi ne basta 1 di segno, eseguiremo quindi uno *shift* verso destra di 15 bit ottenendo ss15; s15 verrà messo negli LSB mentre l'altra "s" rimarrà negli MSB e quindi verrà persa (si ricorda che si lavora a 16 bit)

divisioni: Per quanto riguarda le divisioni, in alcuni processori sono meno veloci delle moltiplicazioni, per velocizzarle si cerca di eseguire il numero maggiore di moltiplicazioni , traforando, quando è possibile, divisioni in frazioni dell'unità: 52/60 può essere vista come 52 x 1/60 ovvero 52 x 0,1667.

Somme e sottrazioni: sono le operazioni che possono provocare l'overflow, si rende necessario in sede di programmazione accertarsi che i risultati siano tra i numeri rappresentabili.

Complemento a 2 (C2): [Mian 99] La notazione in complemento a 2 è il reso non negativo r , della divisione per 2^{n+1}

$$x = q2^{n+1} + r \qquad \begin{cases} q = 0 & 0 \le x \le 2^{n+1} - 1 \\ q = 1 & -2^{n+1} \le x \le 0 \end{cases} \qquad (8.3)$$

Un metodo veloce per calcolarla consiste nel:

- convertire il modulo del numero in base 2 , ci si ferma qui se è positivo. (ad esempio -6→ 0110)
- complementare tutti i bit , (ad esempio 0110→1001)
- sommare 1, , (ad esempio 1001+0001=1010 che rappresenta –6 in C2)

questa notazione ha numerosi vantaggi:

- unicità dello zero
- semplicità di conversione dalla notazione C2 ad *n* bit a quella ad *m* bit con *m*>*n* , che si effettua con una propagazione del segno
- semplicità nelle operazione di somma/differenza di due numeri, che viene effettuata modulo 2^{n+1},, cioè sommando bit per bit le loro rappresentazioni in C2 e trascurando il riporto eventuale alla sinistra del bit più significativo; quindi la somma in c2 corrisponde alla somma ordinaria. L'overflow si può verificare tra numeri dello stesso segno, ed – segnalato dalla presenza di un risultato con segno diverso dagli addendi.
- Modularità della operazione di somma: la caratteristica ingresso-uscita di un sommatore in C2 è periodica. Si dimostra che se *S* (risultato della somma) appartiene ai numeri rappresentabili $S \in [-2^n, +2^n - 1]$, gli eventuali overflow nelle somme parziali non hanno effetto sulla correttezza del risultato finale.

  Come esempio si considerino numeri a 4 bit (3+1 di segno) S appartiene a [-8,7] eseguiamo S=7+5-6 calcolati come ((7+5)-6)

$$\begin{array}{ll} 0111 + & 7 + \\ \underline{0101} = & 5 = \\ 1100 + & 12 + \rightarrow Overflow \\ \underline{1010} = & -6 \\ 0110 \rightarrow & 6 \rightarrow Corretto \end{array} \qquad (8.4)$$

OSSERVAZIONE:La notazione da noi utilizzata è la 0.15, per implementare le funzioni seno e coseno, mentre per l'odometria una specie di virgola mobile, per mantenere l'errore di calcoli i più bassi possibili

8.2 Funzioni *sin* e *cos*

In questo paragrafo, si implementano le funzioni trigonometriche seno e coseno cercando di rendere il calcolo il più veloce possibile, esigenza indispensabile nei sistemi Run Time

8.2.1 Riduzione ad un solo quadrante

Attraverso lo studio della trigonometria si osserva come il seno ed il coseno siano strettamente legati dalla relazione:

$$\cos(x) = \sin\left(\frac{\pi}{2} - x\right) \qquad (8.5)$$

quindi è possibile calcolare solamente il seno e da esso ricavare il coseno.
Un'altra utile osservazione è che ci si può ricondurre a lavorare unicamente nel primo quadrante effettuando semplici operazioni di traslazione di angoli, valgono infatti le relazioni:

$$\begin{array}{llll} \sin(x) & 0 \le x \le \frac{\pi}{2} & \cos(x) = \sin\left(\frac{\pi}{2} - x\right) & 0 \le x \le \frac{\pi}{2} \\ \sin(x) = \sin(\pi - x) & \frac{\pi}{2} \le x \le \pi & \cos(x) = -\sin\left(x - \frac{\pi}{2}\right) & \frac{\pi}{2} \le x \le \pi \\ \sin(x) = -\sin(\pi - x) & \pi \le x \le \frac{3}{2}\pi & \cos(x) = -\sin\left(\frac{3}{2}\pi - x\right) & \pi \le x \le \frac{3}{2}\pi \\ \sin(x) = -\sin(2\pi - x) & \frac{3}{2}\pi \le x \le 2\pi & \cos(x) = \sin\left(x - \frac{3}{2}\pi\right) & \frac{3}{2}\pi \le x \le 2\pi \end{array} \qquad (8.6)$$

A questo punto dato qualsiasi angolo , possiamo riportarci al calcolo di un seno sul primo quadrante , ossia ad un angolo compreso tra 0 e 19° .

8.2.2 Serie di Tylor

La prima idea che può venire in mente è esprimere il seno come sviluppo della serie di Tylor:

$$\sin(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots \qquad (8.7)$$

nel nostro caso si calcolerà una serie troncata ad un punto tale da rendere l'errore di arrotondamento trascurabile, questo accade per n=5-6.
Osservando ogni addendo si nota come sia necessario moltiplicare il precedente per un quadrato e dividere per numeri crescenti:

$$\begin{array}{l} \frac{x^3}{3!} = x \cdot \frac{x^2}{2 \cdot 3} \\ \frac{x^5}{5!} = \frac{x^3}{3!} \cdot \frac{x^2}{4 \cdot 5} \\ \frac{x^7}{7!} = \frac{x^5}{5!} \cdot \frac{x^2}{6 \cdot 7} \end{array} \qquad (8.8)$$

La divisione può essere sostituita con prodotti per frazioni memorizzabili a priori in un array, ma la moltiplicazione implica arrotondamenti. Lo stesso quadrato implica un arrotondamento, quindi si lavora con un numero di per sé già impreciso ed ad ogni moltiplicazione otteniamo un risultato che si allontana sempre più dall'accuratezza da cui siamo partiti.
Il secondo motivo per il quale non si è utilizzato tale metodo risiede grado degli esponenti, degli elevamenti portano spesso il sistema in overflow, specialmente nei primi addenti in cui il denominatore non è elevato

8.2.3 SIN come risposta impulsiva della traformata Zeta

[Oppenheim, Shafer 99] La trasformata zeta unilatera viene definita come:

$$X(z) = \sum_{n=0}^{\infty} x[n] z^{-n} \qquad (8.9)$$

serve per esprimere in modo compatto un segnale campionato, elaborarlo e successivamente ricostruirlo (se il tempo di campionamento era corretto)
la trasformata del seno vale:

$$\sin(\omega_0 n)u[n] \xrightarrow{Z} \frac{[\sin\omega_0]z^{-1}}{1-[2\cos\omega_0]z^{-1}+z^{-2}} \qquad |z|>1 \qquad (8.10)$$

con *u[n]* gradino unitario,*n* è un numero intero ed indica il numero del campione, ω_0 ci permette di definire la frequenza di campionamento, se desideriamo un campione ogni grado $\omega_0 = \frac{\pi}{180}$ (si ricorda che l'argomento del seno viene sempre espresso in radianti). Chiamiamo Hsin(z) la trasformata zeta del seno. Possiamo scrivere:

$$H\sin(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}} = \frac{z b_1}{z^2 - a_1 z - a_2} = q_0 + \frac{s}{z-p} + \frac{s^*}{z-p^*} \qquad (8.11)$$

con "*" si intende complesso e coniugato, sia *s* che *p* sono numeri complessi
nella (8.11) si è scritta la funzione di trasferimento Hsin(z) prima nella forma generale, nel secondo passaggio si sono eliminati i termini nulli ed trasformata in potenze positive di z. l'ultima espressione indica la forma generale di scomposizione per poi poter effettuare l'antitrasformata;

$$\begin{cases} b_0 = 0 \\ b_1 = \sin(\omega_0) \\ b_2 = 0 \end{cases} \qquad \begin{cases} a_0 = 1 \\ a_1 = 2\cos(\omega_0) \\ a_2 = -1 \end{cases} \qquad (8.12)$$

attuando alcuni semplici passaggi si ricava che:

$$q_0 = 0$$

$$p = 1e^{+j\omega_0} \qquad (8.13)$$

$$s = (\alpha + j\beta) = \begin{cases} \alpha = \frac{b_1}{2} = \frac{\sin(\omega_0)}{2} \\ \beta = -\frac{\cos(\omega_0)}{2} \end{cases}$$

Il modulo di *p* uguale significa che il sistema non è stabile e l'uscita (hsin(n)) avrà un moto oscillatorio , ed è quello che ci si aspetta dovendo ricostruire un seno.

Dato il terzo membro della (8.11) posso effettuare l'antitraformmata Zeta, ottenendo la risposta impulsiva di Hsin(z): hsin(n)

$$h\sin(n) = (sp^{n-1} + s^* p^{*n-1})1(n) \qquad (8.14)$$

Attraverso alcuni passaggi è possibile semplificare ancor più la (8.14) ottenendo la funzione seno come a primo membro della 8.10; ma operando così torneremmo al punto di partenza.
Invece stiamo cercando è un metodo iterativo che ci permetta di costruire la funzione seno, l'abbiamo trovato nella (8.14). infatti

$$\sin(x) = \sum_{n=0}^{x} h\sin(n) = \sum_{n-0}^{x} (sp^{n-1} + s^* p^{*n-1}) \qquad (8.15)$$

NOTA: *x* è un angolo intero ESPRESSO IN GRADI

Si è scelto di lavorare con angoli espressi per facilità di rappresentazione dal punto di vista implementativo[70], ed anche per rendere la lettura del sensore di rotazione più agevole, (ogni *hole* contata corrisponde ad 1° spaziato)

Si potrebbe obiettare che anche in questa formula vi sono potenze da calcolare miste a somme, ma in questo caso è possibile realizzare i prodotti in modo ricorsivo. Infatti calcolando i primi termini si nota:
Indichiamo con

$$p = \alpha + j\beta \qquad s = a + jb \qquad (8.16)$$

(ATTENZIONE la notazione è diversa da quella di (8.13))

$$\begin{aligned} h\sin(1) &= s + s^* = (a + jb) + (a - jb) = \ldots = 2a \\ h\sin(2) &= (a + jb)(\alpha + j\beta) + (a - jb)(\alpha - j\beta) = \ldots = 2(a\alpha - b\beta) \\ h\sin(3) &= (a + jb)(\alpha + j\beta)^2 + (a - jb)(\alpha - j\beta)^2 = \ldots = 2[a(\alpha^2 - \beta^2) - b(2\alpha\beta)] \end{aligned} \qquad (8.17)$$

Osservando attentamente i risultati si può ricavare un andamento ripetitivo:

$$h\sin(1) = 2a$$

$$h\sin(2) = 2(a\alpha^I - b\beta^I) \qquad \begin{cases} \alpha^I = \alpha \\ \beta^I = \beta \end{cases}$$

$$h\sin(3) = 2(a\alpha^{II} - b\beta^{II}) \qquad \begin{cases} \alpha^{II} = \alpha\alpha^I - \beta\beta^I \\ \beta^{II} = \alpha^I\beta + a\beta^I \end{cases} \qquad (8.18)$$

$$h\sin(4) = 2(a\alpha^{III} - b\beta^{III}) \qquad \begin{cases} \alpha^{III} = \alpha^I\alpha^{II} - \beta^I\beta^{II} \\ \beta^{III} = \alpha^I\beta^{II} + \beta^I\alpha^{II} \end{cases}$$

In definitiva la formula generale è

$$h\sin(n) = 2(a\alpha^{N-1} - b\beta^{N-1}) \qquad \begin{cases} \alpha^{N-1} = \alpha^I\alpha^{N-2} - \beta^I\beta^{N-2} \\ \beta^{N-1} = \alpha^I\beta^{N-2} + \beta^I\alpha^{N-2} \end{cases} \qquad (8.19)$$

[70] Si ricordi le formule dell'odometria in cui $\varphi_{\deg} = \Delta\alpha \cdot 180$

Gli aspetti positivi di questa formula consistono in :

- Non si effettuano mai divisioni
- Sono necessarie solamente 4 variabili moltiplicate a due a due e poi sottratte, la struttura base di *hsin* è sempre costante
- Non si lavora con numeri complessi
- Ad ogni iterazione i valori delle variabili $\alpha^{N-1}, \beta^{N-1}$ vengono calcolati dai precedenti (N-2) moltiplicandoli per delle costanti: infatti α, β si conoscono a priori
- Non vi sono problemi di overflow, il fondo scale è al massimo uno.

gli aspetti negativi consistono in

- Ricordarsi che la funzione seno implementata accetta come ingresso un angolo in Gradi
- Il tempo di calcolo non è costante, aumenta all'aumentare del valore dell'angolo

Sorgente C

DI seguito un estratto del codice dal file *includ_math.c*in cui vengono inizializzate le 4 variabili iniziali, la parte reale ed immaginaria di *p e s ,* e poi viene implementata la funzione seno.

È interessante osservare anche i metodi di arrotondamento in cui si somma un 1 al bit che dopo lo *shift* destro (di 14 bit) diverrà il meno significativo

```
/*! \file   include_math.c
    \author Maragno Simone <simomail@inwind.it>
*/

#include <conio.h>
#include <../programmi/seno.dat.c>

////////////////////////////////////////////////////////////////////////////
//
// Global variable
//
////////////////////////////////////////////////////////////////////////////
long sin( int);
long cos(int);

const unsigned long int    pitrue=0x3243f;// 3.141586304  errore di 0.000006349

//const unsigned int pi=0xc90f;          //.2.14       error=0.000052

const unsigned int pi=0x6487, //s.2.13       error=0.000113
Re_p=0x7ffb,              //cos(wo)=0.99984741210938
Im_p=0x23c,               //sin(wo)=0.01745605468750
Re_scos=0x3ffe,           //cos(wo)/2=0.49993896484375
Im_scos=0x11e,            //sin(wo)/2=0.00872802734375
Re_ssin=0x11e,             //sin(wo)/2
Im_ssin=0xc002;            //-cos(wo)/2


const int wo=0x23b           ;                              // wo=pi/180


////////////////////////////////////////////////////////////////////////////
//
// Functions
//
```

```
////////////////////////////////////////////////////////////////////////////

/*************************************************************************
*  CALCOLO SENO IN MODO ITERATIVO
*
* Utile quanto si ha poca memoria a disp.
*
*     Dalla TRASF. ZETA di seno e coseno decompongo in
*                       H(z)=b+   s1      + s2
*                                         -----  -----
*                                       z-p1   z-p2
*     con s1 e s2, p1 e p2  complessi e coniugati
*     b=1 per coseno, b=0 per seno
*     Costanti quantizzate a 16 bit (15+segno)
*
* calcoliamo la risposta impulsiva dell'anti trasf. Z
*      h(n)=[s1*p1^(n-1)+s2*p2^(n-1)]1(n)
* nel caso del coseno si aggiunge "+delta(n)"
* Si ricava:   h(n)=2*(Re_s*Re_new-Im_s*Im_new)
* con
*     Re_new=Re_old*Re_p-Im_old*Im_p
*     Im_new=Re_p*Im_old+Im_p*Re_old
*************************************************************************/

int sin( int x){
      long int tmp,hsin=Re_ssin,Re_old=Re_p,Im_old=Im_p,Re_new,Im_new;
      int n;
      if(x==0) return (hsin=0);                  //sin(0)=0
      if(x==1) return (hsin=2*Re_ssin);          //sin(1)=2a
      if(x==2) {
            hsin*=Re_old;
            tmp=-Im_ssin;
            tmp*=Im_old;
            hsin+=tmp;
            hsin+=1<<14;                        //equiv a:  hsin*=2;
            hsin+=1<<15;                        //arrotondo
            return (hsin>>14);                  // shifto nell'LSB
       }
      for(n=3;n<=x;n++){
       Re_new=Re_old;
            Re_new*=Re_p;
            tmp=-Im_old;
            tmp*=Im_p;
            Re_new+=tmp;      //non arrotondo
            Re_new=Re_new>>15;
   Im_new=Re_p;
      Im_new*=Im_old;
      tmp=Im_p;
      tmp*=Re_old;
      Im_new+=tmp;
      Im_new=Im_new>>15;
   hsin=Re_ssin;
    hsin*=Re_new;
    tmp=-Im_ssin;
    tmp*=Im_new;
    hsin+=tmp+0x2000;   // +1<<14
    Re_old=Re_new;
    Im_old=Im_new;
    //cputw(hsin>>14);
    //sleep(3);
   }
  return (hsin>>14);
```

```
}
```

8.2.4 SIN come array

Lo scorso paragrafo è stato introdotto un metodo che minimizza l'occupazione dello spazio in memoria per il calcolo del seno, se i nostri programmi non sono così lunghi ed abbiamo disponibilità di spazio, è possibile utilizzare un metodo molto semplice: un Array
Viene creato un array in cui vengono scritti tutti i valori del seno, quatizzati a 16 bit,per angoli espressi in gradi da 0° a 90° , successivamente si crea un programma che applicando le proprietà viste al paragrafo 8.2.1, restituisca il seno di qualsiasi angolo (positivo, negativo, maggiore di 360°,..) purché espresso in gradi
I vantaggi di tale metodo sono:

- Alta velocità di esecuzione
- Accuratezza, (non ci sono arrotondamenti), esprimo i valori quantizzandoli a 16 bit (è anche la massima accuratezza raggiungibile con questo microcontrollore)

Gli svantaggi:

- Creo un array con 91[71] valori di tipo `unsigned int` (si ricorda che la memoria disonibile per programmi sono circa 12k byte)

Estratto C

Nel file *seno.dat.c* è presente l'array, creato con l'instruzione Matlab
`sinq=round(2^15*sin(0:90))`
Di seguito un estratto dal file *include_math.c* in cui vengono implementate 2 funzioni, una per il seno e l'altra per il calcolo del coseno.
Infine è presente anche un programma di test, che lancia le 2 funzioni e stampa i risultati sull'LCD

[71] da 0° a 90° compresi

```
/*! \file   include_math.c
    \author Maragno Simone <simomail@inwind.it>
*/

#include <conio.h>
#include <../programmi/seno.dat.c>

////////////////////////////////////////////////////////////////////////////
/
//
// Global variable
//
////////////////////////////////////////////////////////////////////////////
/
long sin( int);
long cos(int);

const unsigned long int    pitrue=0x3243f;// 3.141586304  errore di
0.000006349
//const unsigned int pi=0xc90f;           //.2.14       error=0.000052

const unsigned int pi=0x6487, //s.2.13       error=0.000113

////////////////////////////////////////////////////////////////////////////
/
//
// Functions
//
////////////////////////////////////////////////////////////////////////////

/*************************************************************************
*
*                                    CALCOLO SENO di X ( in gradi)
* sfruttando Array(0-90) in cui sono memoriz. i valori del seno quantizzati
* a 16 bit.
* Estratto matlab:                   sinq=round(2^15*sin(0:90))
*************************************************************************
*/
long sin(int x){
      int y,s=0;
      long o=-1;
      if(x<0){s=1,x*=-1;}
      if (x>360) {
            y=x/0x168;
            y*=0x168;
            x-=y;
            }
      if(s==1)x=360-x;
      if (x<=90) return (sinx[x]);
      else  if (x<=180) return (sinx[180-x]);
                        else  if(x<=270) return (long)(o*=sinx[x-180]);
                                         else return(o*=sinx[360-x]);
 }

/*************************************************************************
*
*                             CALCOLO COSENO di X ( in gradi)
*sfruttando la relazione   cos(x)=sen(90-x)
*************************************************************************
/
long cos(int x){
      int y,s=0;
```

```
      long o=-1;
      if(x<0){s=1,x*=-1;}
      if (x>360) {
            y=x/0x168;
            y*=0x168;
            x-=y;
            }
      if(s==1)x=360-x;
      if (x<=90) return (sinx[90-x]);
      else  if (x<=180) return (o*=sinx[x-90]);
                        else  if(x<=270) return (o*=sinx[270-x]);
                                         else return(sinx[x-270]);
 }


/*****************************************************************************
*
*                              TEST PROGRAM
*****************************************************************************
*/
/*int  main(int argc, char *argv[]){
      int i=0,s,* ang[4];
      long conv;
      ang[0]=359 ;
      ang[1]=-90;
      ang[2]=-840;
      ang[3]=260;
      while(1){
            //conv=sin(i++);
            conv=sin(ang[i++]);
            //s=conv>>16;
            //conv=cos(ang[i++]);
            conv*=0x2710;                       // molt *10000
            //conv*=0x3e8;                      //molt *1000   per angolo 90
                                                // altrimenti vado fuori LCD
            conv=conv/0x8000;   //dividendo per 2^15   ottengo  numero in
                                  decimale da 0.XXXX a XXXX*E-4;
            //cputw(s);
            //sleep(1);
            lcd_int(conv);                //nel caso di 1.0000 5 digit non ci
stanno.
            sleep(3);
            if(i==4) return 0;
      }
}
*/
```

Capitolo 9 Sensor multiplexing

Sensor multiplexing

Avendo a disposizione più sensori del numero di porte per contenerli, è necessario l'utilizzo di un sistema di multiplexing delle porte di ingresso. In particolare:

- PORTA 1: sensore di rotazione 1, utilizzato per l'odometria del motore A
- PORTA 3 : sensore di rotazione 3, utilizzato per l'odometria del motore B
- PORTA 2:
  - Light sensor (per il proximity sensor)
  - Bump sensor 1
  - Bump sensor 2

Vi sono varie soluzione che prevedono di sfruttare la sensoristica già disponibile, altre l'utilizzo di circuiteria esterna, infine la multiplazioni di due nuovi sensori costruiti su di uno stesso circuito esterno.

9.1 Architetture lego

Scartiamo a priori la possibilità di utilizzare solamente il Light sensor perché come visto nel Cap. 4 , il proximity sensor permetterebbe di seguire solamente muri convessi ed ostacoli frontali non verrebbero rilevati.

9.1.1 Serie: Bump AND

[Gasperi 98] La prima modifica che può venire in mente è l'utilizzo di due bump sensor su di un'unica porta di input, come si nota in Figura 9.1.

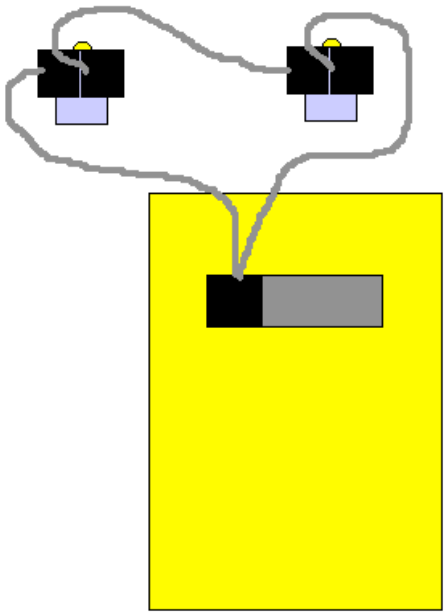


Figura 9.1 bump AND

I due sensori sono stati collegati in serie, l'uscita sarà ad uno soalemnte quando entrambi vengono premuti, è stata creata una porta AND.
L'aspetto negativo di questa soluzione, anche lavorando con una logica inversa, come quella vista al Cap 4, è il non poter riconoscere quale bump è stato premuto, per tale ragione non è stato utilizzato.

9.1.2 Light & Bump

Parallelo
Una tecnica simile alla precedente consiste nel collegare alla stessa porta sia un bump sia il light sensor., ma non in serie come nel paragrafo 9.1.1 ma bensì in parallelo.
Dei due l'unico a dover essere alimentato è il light, il bump finchè non viene premuto non altera la tensione ai suoi capi, quando invece viene sollecitato l'uscita assume un valore basso. In questo caso non si può lavorare in logica inversa, con il bump sempre premuto, perché equivale a mettere in corto l'alimentazione del light, rilevando sempre un valore di1023.
Se si lavora nella normale configurazione, è possibile accorgersi della pressione del touch sensor dal brusco abbassamento dei valori rilevati.
Gli aspetti negativi di tale modalità sono:

- Rielaborare un architettura meccanica per il bump sensor, in logica non inversa, che lavori altrettanto bene di quella utilizzata fin qui.
  Non è ammissibile una involuzione operativa del robot.
- L'insorgere di oscillazioni dovute alla chiusura dell'interruttore contenuto nel bump, si ricorda che quando il canale di input veniva inizializzato come touch sensor, i valori venivano rilevati dopo un tempo di assestamento di 300ms

Serie
È possibile collegare i due sensori , in modo simile al paragrafo 9.1.1, con il bump in configurazione inversa, sempre premuto, in modo da permettere l'alimentazione del light.
Quando il bump sensor viene sollecitato , l'interruttore al suo interno si apre e non passa più corrente, l'alimentazione al light sensor viene sospesa ed il valore rilevato è quello massimo (1023).
I problemi in questa configurazione sono:

- L'insorgere di oscillazioni dovute alla apertura dell'interruttore
- Errato rilevamento del proximity sensor, si ricorda che il segnale IR viene letto dal light sensor quando si presenta un aumento nella rilevazione di più di 200 (valore quantizzato) e 1023 è sicuramente maggiore di tale valore. Per ovviare a ciò si dovrebbero fissare due soglie superiori, ma questo vuol dire trovare il massimo valore rilevabile dal light sensor quando è irradiato da luce infrarossa.
  Questo valore è difficilmente calcolabile, dipende dalla luce ambientale e dalla distanza del robot da un ostacolo

9.2 Bump multiplexing

La soluzione che segue per ora permette solamente il multiplexing di bump sensor in quanto non riesce a fornire l'alimentazione ad altri sensori, affinché ciò accadesse sarebbe stato necessario modificare il light sensor bypassando al suo interno tutta l'interfaccia per i sensori, oppure riprogettandolo.

9.2.1 Schema elettrico

Il dispositivo utilizzato è quello presente in Figura 9.2, permette lo switch tra i diversi ingressi (Input1…Input3). Si suppone che tale dispositivo venga connesso ad una porta di ingresso dell'RCX e questa venga configurata come Light sensor a lettura percentuale.
Si ricorda che 0volt equivalgono a 100 e 5volt a 0
In tal caso verrà rilevato:

- 0 → sensore aperto
- 88→ sensore premuto

Analizziamo il circuito in dettaglio:

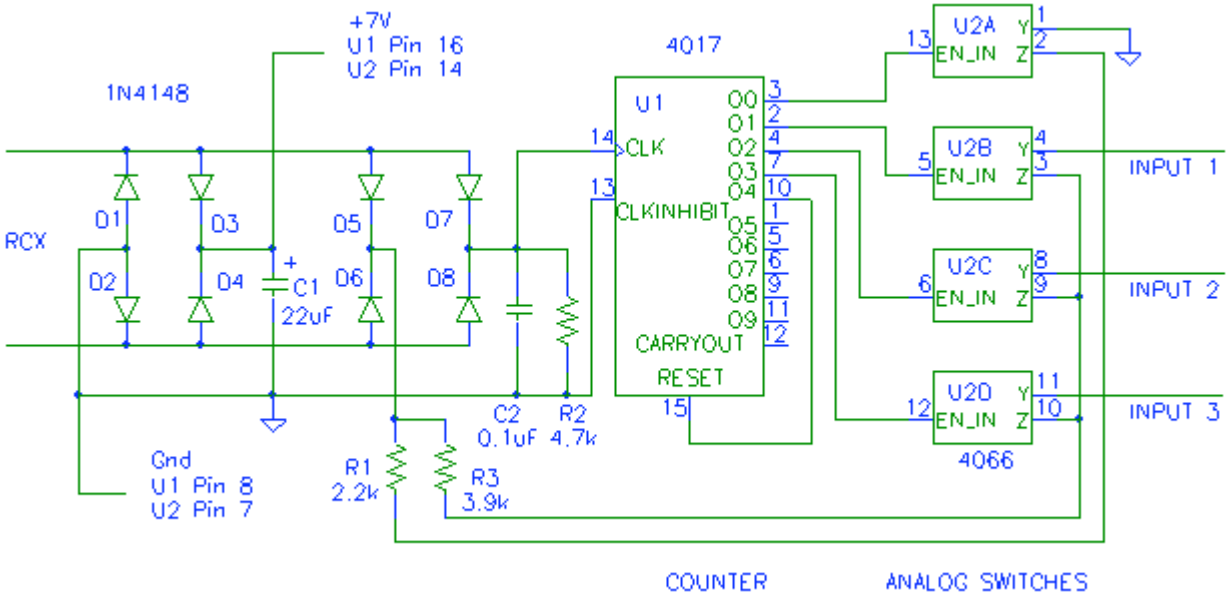


Figura 9.2 Bump multiplexing

`Diodi D1,D2,D3,D4,C1:` Permettono l'interfaccia con le porte di ingresso, servono per alimentare il circuito, in particolare lo Switch analogico ed il Contatore. Il condensatore viene utilizzato per mantenere costante la tensione di alimentazione anche quando avviene il rilevamento da parte dell'A/D

`Diodi D7,D8,C2,R2:` Attraverso lo il cambiamento tra sensore definito come light sensor e touch sensor (in NQC), ovvero attraverso l'attivazione dell'limentazione e della sospensione della stessa, viene creato il clock per in counter 4017

`Contatore Johnson 4017:`(integrato a16 pin) .È un contatore a 5 stadi :

- tipica frequenza di conteggio 13.8MHz a Vdd=10volt
- decodifica delle uscite attiva alta
- conteggio settabile tre transizioni alto-basso o basso-alto
- possibilita di collegamento in cascata

Nella configurazione di figura il conteggio avviene quando il pin del Reset (15) e il 13 sono bassi e avviene una transizione basso-alta del pin del clock (14)
Ad ogni pulsazione del clock vengono attivate in sequenza le uscite 00, 01, 02, 03, 04, quest'ultima attiva il reset del conteggio. Al prossimo clock si reinizierà da 00

`Switch 4066:` Motorola Mc14066, (14pin). È un integrato costituito da 4 switch indipendenti , è capace di controllare sia segnali analogici che digitali. Ogni switch può essere visto come un interruttore, "Y, Z" costituiscono i segnali ai capi dell'interruttore

mentre i segnali ai piedini “13, 5, 6, 12” costituiscono i comandi che chiudono gli interruttori.
Le caratteristiche di questo integrato sono:

- Alto rapporto di tensione On/Off –65dB
- Quiescent current = 0.5nA/blocco a 5Vdc
- Range della tensione di alimentazione da 3 Vdc a 18 Vdc
- Protezione a diodi su ogni ingresso
- Frequenza di trasmissione di 65MHz a 10 Vdc
- Basso rumore –12nv/$\sqrt{ciclo}$, per f > 1Hz

Si sono utilizzati 4 switch perché il primo viene attivato dal primo valore del counter è zero, sicollega quindi l’igresso Y dello switch U2A a massa proprio per individuare quando inizia il conteggio grazie alla resistenza R3 , gli altri switch sono connessi a tre ingressi, noi ne abbiamo utilizzato 2, avendo solamente 2 bump sensor.

D5,D6: Interfaccia per le porte di ingresso, servono per istradare i valori dei sensori) (INPUT1..INPUT3) attraverso le resistenze R1 e R3

R3: Permette di individuare quando inizia il conteggio del counter, infatti quando ciò accade attraverso lo switch U2A, viene collegata a massa e l’RCX legge 100 (se configurato come sensore di luminosità), questo è il valore massimo rlevabile

R1: Essendo di valore più grande di R3 il segnale rilevato dall’RCX non supererà mai il 91 (valore rilevato per ingresso nullo)

È possibile espandere il numero dei canali aggiungendo un altro 4066 tra le uscite 04..07 del contatore, posizionando il RESET in 08

9.2.2 Architettura meccanica

La basetta del circuito è grande 4.4cm x 2.2cm, si è ritenuto opportuno posizionarla sul fondo del RCX , in modo da occupare poco spazio come si vede in figura Figura 9.3, viene incastrata sotto il supporto per l'asse delle ruote.
Nella foto di destra , in alto è possibile notare come i cavi (rosso, nero) che vanno alle porte dell’RCX vengano inseriti sui supporti di collegamento dei sensori, questi sono costituiti da una coppia di mattoncini lego 2x2 collegati da un cavo, sulla cui sommità al posto delle classiche cunette di incastro sono presenti dei contatti metallici, mentre nella parte inferiore, sui lati interni, vi sono delle lamine metalliche. I fili del multiplexer vengono incastrati nei connettori lego tra la plastica e la lamina.
In Figura 9.4 è ancora più evidente ciò, si vedono in alto 3 connettori collegati ad un mattoncino grigio di supporto, si tratta di un mattoncino di quelli forati); sotto del quale è possibile scorgere i cavi che arrivano dal multipexer e terminano con una barretta in rame, (quella che andrà ad incastrarsi).
Questi 3 connettori sono i collegamenti ai tra Input visti nello schema del circuito.

Tale metodo di connessione ha evitato la necessità di costruire dei connettori fatti in casa, come è accaduto per il sensore di rotazione, i cui connettori si possono vedere in Figura 9.5 .Dove sono stati irrimediabilmente modificati dei pezzi lego, e si possono scorgere nitidamente le lamine di metallo all’interno che permettono il collegamento con i contatti metalli sull'RCX

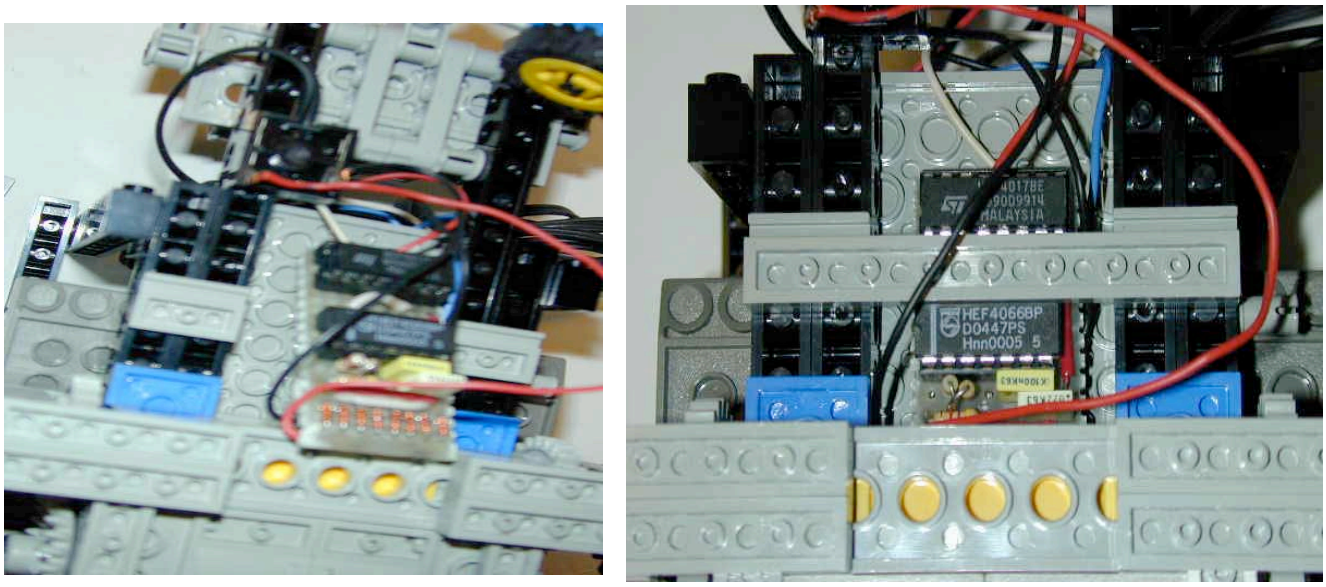


Figura 9.3 Posizionamento scheda Multiplexer

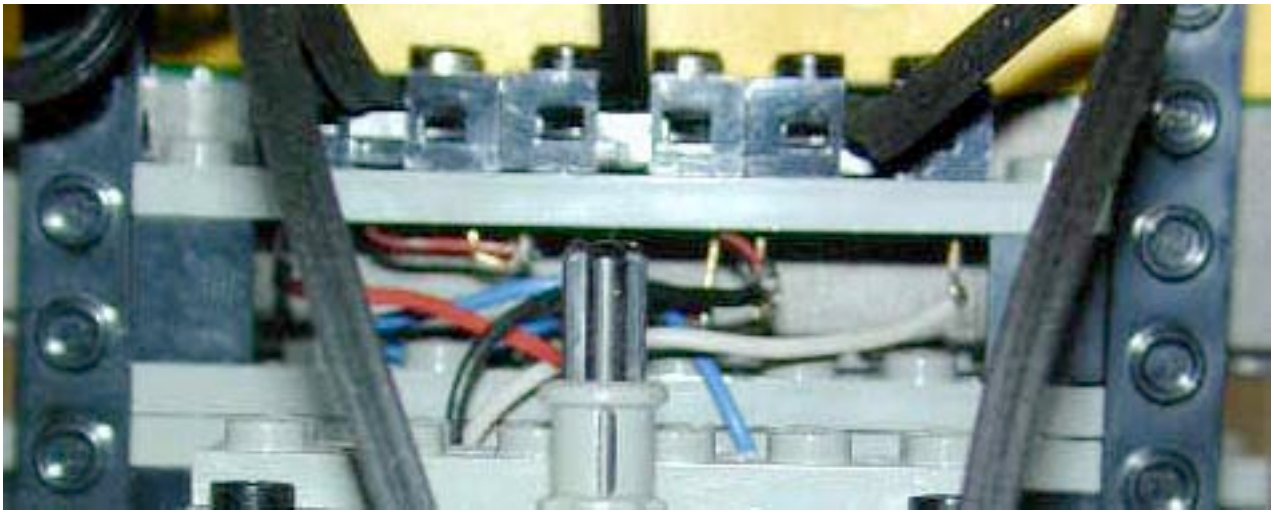


Figura 9.4 collegamenti ai connettori dei sensori

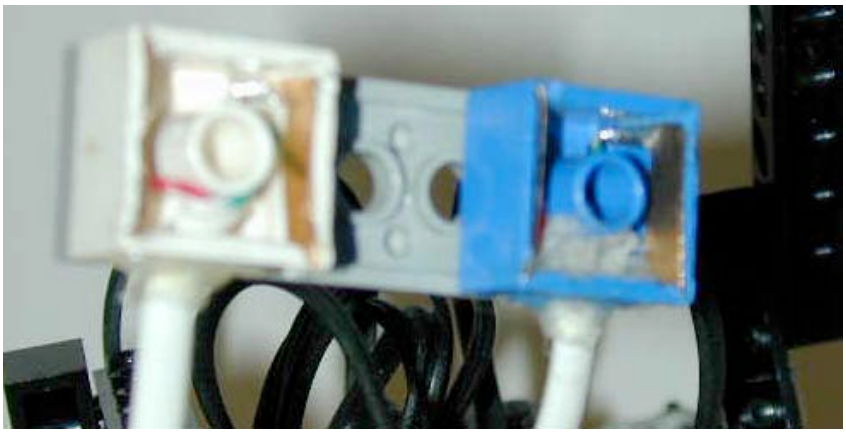


Figura 9.5 Connettori del sensore di rotazione

9.2.3 Implementazione software

Questo circuito è stato inizialmente gestito in NQC , la trasposizione in C è di facile realizzazione con l'accortezza di ricordarsi che in NQC definire un sensore definito come Light attiva l'alimentazione alla porta di input considerata, mentre in uno definito come Touch non viene alimentata la porta. Nel legoS invece, definire uno o l'altro non cambia in quanto vi è la possibilità di leggere dal light sensor in modo passivo senza accendere il led rosso , quindi senza alimentarlo. L'alimentazione viene attivata/disattivata dalle istruzioni:

```
ds_active(&SENSOR_X)
ds_passive(&SENSOR_X)
```

Si ricorda che valore light di 100 descritto in 9.2.1, corrisponde ad un valore quantizzato di 322

All'inizio viene effettuato un ciclo per individuare quando parte il conteggio, rilevando il valore 322, poi, parte un altro ciclo che alle variabili "a,b,c " i valori degli input 1,2,3.

Di seguito viene riportato un estratto di codice C

```
/*! \file   mux_sensor.c
    \brief  test program for mux
    \author Maragno Simone <simomail@inwind.it>
*/

#define sens_2  (int) ds_scale(SENSOR_2)  //valore quantizzato

void main (){
    int a,b,c;
    ds_active(&SENSOR_2);
    while(SENSOR_2 !=322){          //100% light
      ds_passive(&SENSOR_2);
      ds_active(&SENSOR_2);
      msleep(100);
      }
    while(1){
      ds_passive(&SENSOR_2);
      ds_active(&SENSOR_2);
      msleep(100);
      a=sens_2;
      ds_passive(&SENSOR_2);
      ds_active(&SENSOR_2);
      msleep(100);
      b=sens_2;
      ds_passive(&SENSOR_2);
      ds_active(&SENSOR_2);
      msleep(100);
      c=sens_2;
      ds_passive(&SENSOR_2);
      ds_active(&SENSOR_2);
      msleep(100);
    //lcd_int(sens_2);
      }
 }
```

9.3 Infra Red Proximity Detector

[Clark 00] L'ultima soluzione adottata è la creazione di un nuovo sensore che in una sola struttura racchiude i 2 *bump sensor* ed il *proximity sensor,* lasciando libero da utilizzi il tramettitore infrarosso dell'RCX .

9.3.1 Schema elettrico

Lo schema elettrico è quello illustrato in Figura 9.6, assorbe solamente 2.5ma di corrente.

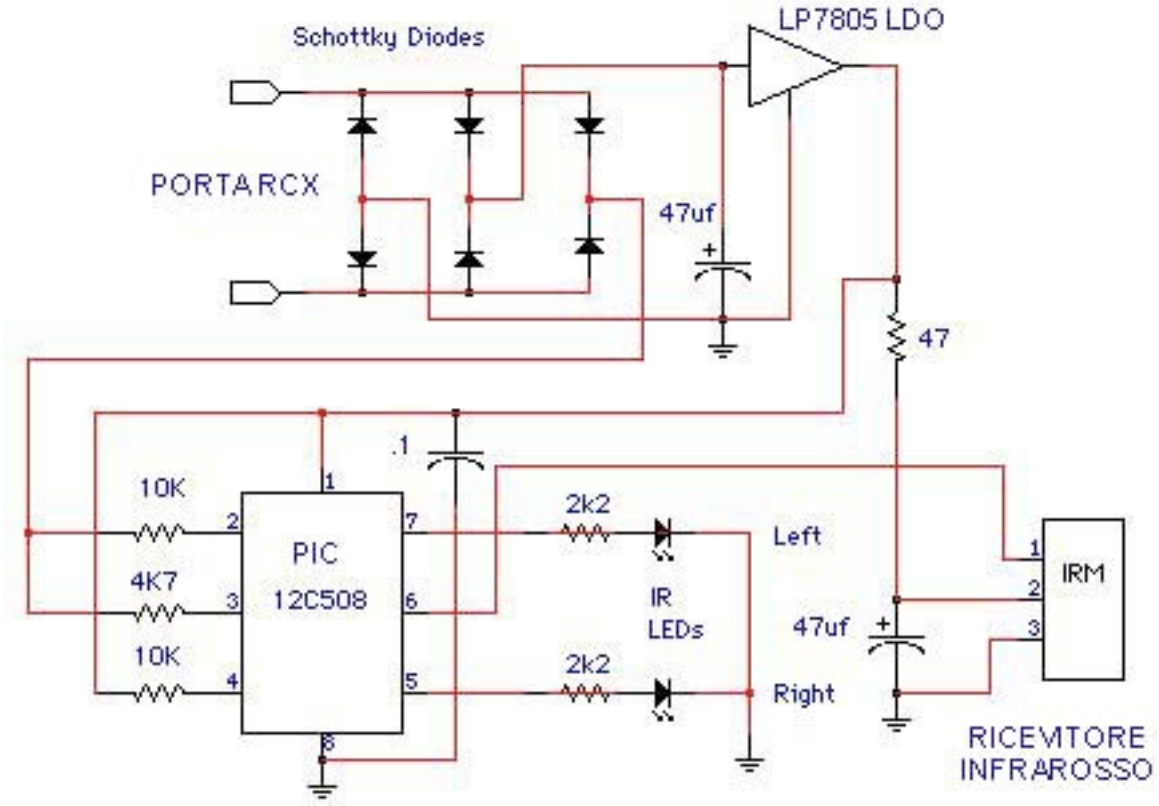


Figura 9.6 circuito dell'IRPD

`Diodi schottky e condensatore`: Come in ogni altra applicazione servono da interfaccia per la porta di ingresso ed il condensatore per stabilizzare la tensione di alimetazione

`LP7805`: È uno stabilizzatore di tensione a 5volt, fa in modo che la tensione in uscita sia costante dovendo alimentare il Pic ed il ricevitore infrarosso

`PIC 12C508`: È un Microchip PICmicro® microcontroller programmabile a 8 pin, si occupa di modulare il segnale trasmesso dai led e di demodulare il segnale ricevuto dal ricevitore infrarosso

Le sue cratteristiche sono:

- Facile da programmare, solo 33 singole istruzioni
- 512x12 word di memoria per il programma
- 25 bytes di RAM
- 4MHz velocità massima
- 4 oscillatori interni
- 2 digital timer
- 6 porte di I/O

`IR led`: Sono utilizzati come trasmettitori Infrarossi, sono contornati da del nastro per focalizzare il raggio di trasmissione altrimenti irradierebbero in tutto lo spazio. Una volta saldati vengono inclinati lateralmente di 5° per permettere il riconoscimento di oggetti laterali.

`IRM`: piccolo ricevitore infrarosso Panasonic a 38kHz , le stesse frequenze dei telecomandi, molto sensibile.

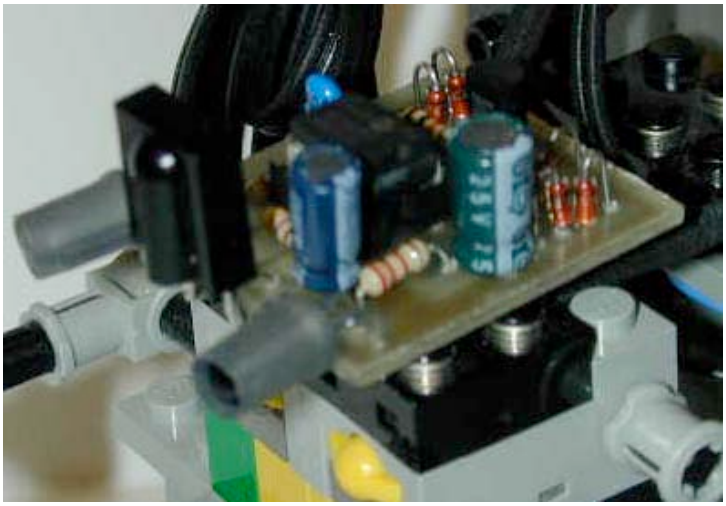


Figura 9.7 IRPD

Il circuito saldato è poco più grande di una moneta da 100 lire, quindi molto compatto, viene inserito nella parte frontale del robot.

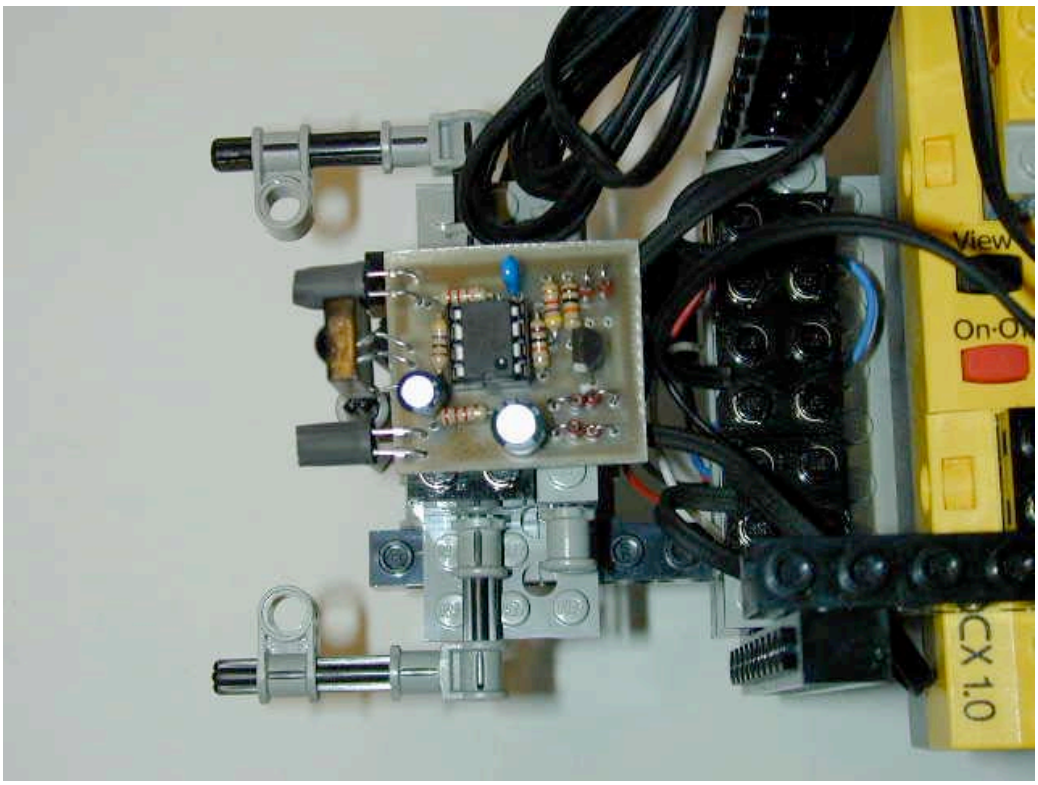


Figura 9.8 IRPD montato sull'RCX

Dalle figure sono evidenti il PIC centrale, i due IR led a destra ed in mezzo a loro il ricevitore infrarosso

9.3.2 Funzionamento

Più precisamente il PIC modula i segnali dei led con impulsi di 40KHz per 2.5ms seguiti da 2.5ms di Off, con un *duty cycle* del 50%. La lettura dal ricevitore avviene nei tempi di Off , in cui non c'è trasmissione di segnale, con la stessa frequenza di 40KHz , vengono quindi effettuati 100 campioni per ciclo.
Terminato un ciclo completo di on/off una lettura è considerata buona se vi sono molte buone rilevazioni, falsa se vi sono meno di 20 buone rilevazioni. Un counter (*good*) viene incrementato ad ogni buon rilevamento, e contemporaneamente un altro counter (*miss*) viene decrementato. Ad ogni cattivo rilevamento le due operazioni si invertono.
Se il *good* counter arriva ad una Soglia stabilita allora viene segnalato il rilevamento e il *miss* counter viene a azzerato; viceversa se il *miss* supera la soglia diventando troppo grande, l'operazione di detect viene disabilitato e il *good* counter viene azzerato.
Con tale sistema i rilevamenti sono più efficaci e la reiezione ai disturbi è migliore
Una volta programmato il PIC, (è necessario un programmatore di Eprom), questo circuito, come enunciato ad inizio paragrafo permette di rilevare ostacoli frontali, a destra, e a sinistra del robot a seconda del valore rilevato.
La distanza i rilevamento è variabile e spazia da 30cm in un luogo ben illuminato ad infinito, tutta la stanza, in un ambiente buio, ovvero viene rilevato il muro opposto.
Il valore rilevato è anche sensibile allo stato delle batterie, ma effettuando delle misurazioni è possibile ottenere i seguenti risultati (Tabella 9.1)

situazione	Valore rilevato (%)
No detection	97-100
Right detection	57-72
Left detection	15-27
Both detection	0-1

Tabella 9.1 IRPD detection

Si nota come possono essere individuate 4 zone in cui il rilevamento non è ambiguo, non essendoci zone di intersezioni, o valori molto vicini tra loro.

9.3.3 Programmazione

Viene di seguito riportato un esempio di codice in NQC per l'utilizzo del IRPD

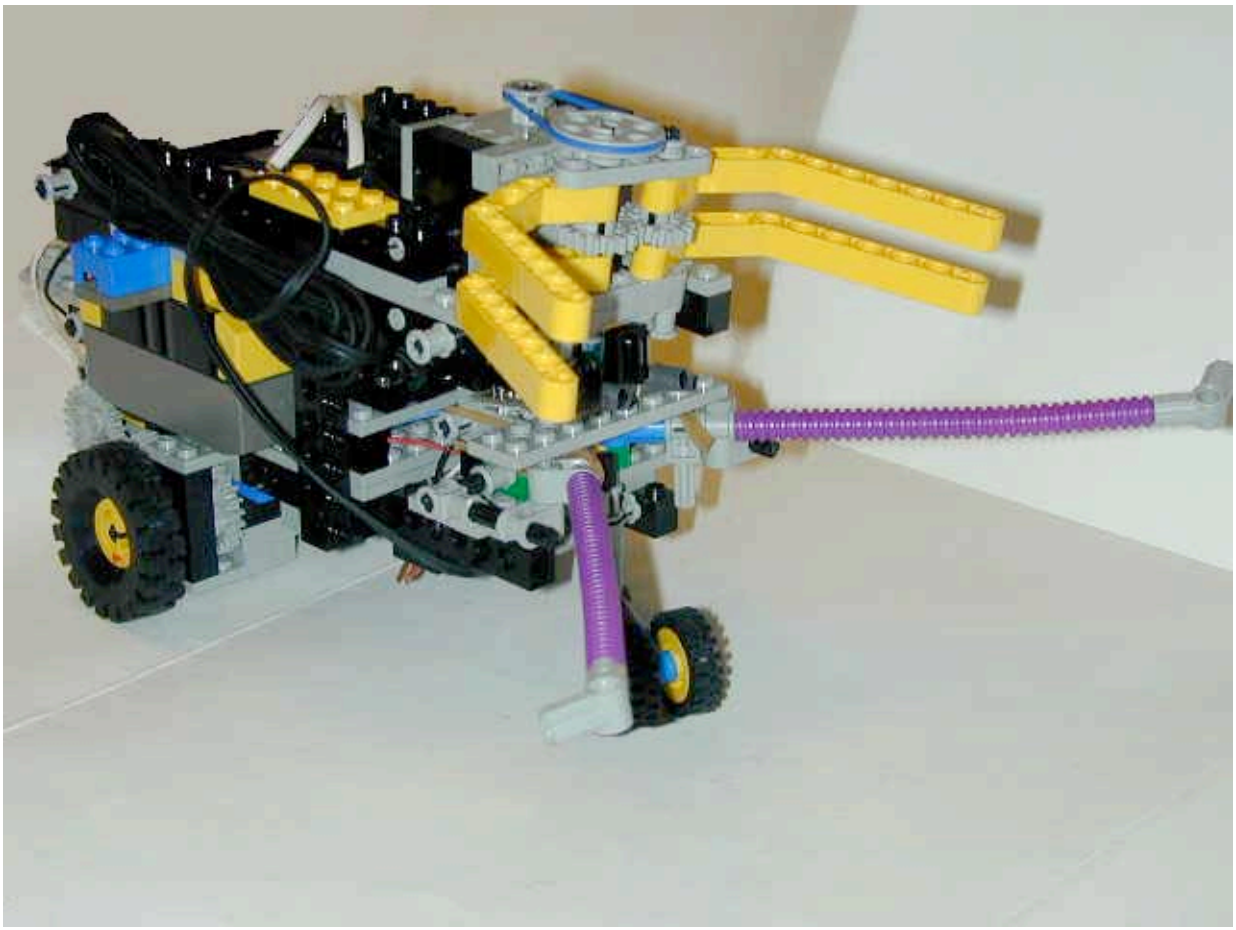
```
#define EYE IN_2
#define LEFT OUT_C
#define RIGHT OUT_A
#define NORMAL_SPEED 10
#define TURN_SPEED 5
#define DELAY 20
#define LBOTH 5
#define LNONE 90
#define LLEFT 57
#define LRIGHT 15

int eye;

task main
{
  Sensor(IN_2, IN_CFG(STYPE_LIGHT, SMODE_PERCENT));
```

```
  Fwd(RIGHT+LEFT, NORMAL_SPEED);
  while(true)
  {
    eye = EYE;
    if (eye < LBOTH)
    {
      Rev(LEFT + RIGHT, NORMAL_SPEED);
      Sleep(100);
      //wait(EYE == LNONE);
      Fwd(LEFT,TURN_SPEED);
      Rev(RIGHT,TURN_SPEED);
      Sleep(DELAY);
      }
    else if (eye < LLEFT && eye > LRIGHT)
    {
      Fwd(LEFT, TURN_SPEED);
      Rev(RIGHT, TURN_SPEED);
      Sleep(DELAY);
     // wait(EYE == LNONE);
     }
   else if (eye > LLEFT && eye < LNONE)
   {
     Fwd(RIGHT, TURN_SPEED);
     Rev(LEFT, TURN_SPEED);
     Sleep(DELAY);
     // wait(EYE == LNONE);
     }
   else
   Fwd(RIGHT+LEFT, NORMAL_SPEED);
 }
}
```

FOTO CON TUTTI I SENSORI



Capitolo 10 Coordinamento di Behavior con parallel task

Coordinamento di Behavior con parallel task

In questo capitolo vengono brevemente esposti quali problemi è possibile incontrare nel coordinare task che vengono eseguito in modo parallelo ovvero in multitasking e che accedono a risorse comuni, i motori
Si studia dapprima la differenza tra fermarli ed interromperli, utilizzando i semafori, e si conclude osservando un coordinamento 'time sharing" ed definendo una coda di priorità

10.1 Parallel task

Sappiamo che sia in NQC che nel legOS è possibile lanciare task che vengono eseguiti in modo parallelo , si ricorda inoltre che non disponendo di processori paralleli, il multitasking è di tipo "*time sharing*" : ogni task viene eseguito per un quanto di tempo.
Prendiamo ad esempio un programma con 2 *task*. Il primo fa eseguire al robot (in configurazione tank) una traiettoria quadrata, percorrendo ogni lato in 1 secondo.
Il secondo *task* rileva la condizione del bump_sensor quando è premuto torna indietro per 5 secondi gira a destra.

Di seguito il cogente NQC:

```
task main()
{
  SetSensor(SENSOR_1,SENSOR_TOUCH);
  start check_sensors;
  while (true)
  {
    OnFwd(OUT_A+OUT_C); Wait(100);
    OnRev(OUT_C); Wait(85);
  }
}

task check_sensors()
{
  while (true)
  {
    if (SENSOR_1 == 1)
    {
      OnRev(OUT_A+OUT_C);
      Wait(50);
      OnFwd(OUT_A);
      Wait(85);
      OnFwd(OUT_C);
    }
  }
}
```

Questo probabilmente potrebbe sembrare un programma valido, ma se lo si esegue i behavior non fanno ciò che desideriamo. Se il robot urta qualcosa mentre sta girando, questo comincia ad andare indietro ma immediatamente muove ancora avanti urtando l'ostacolo.
La ragione di tale comportamento è che i task interferiscono:
L'errore sta nel fatto che quando il secondo task dorme il primo sta ancora funzionando

Task main **Task check_sensor**

	Turn right	sleeping
Sensor=1	sleepping	Going backwards
	Forward ➔ Collisione	sleeping ➔ Non nota la collisione

10.2 Stopping and restarting task

Un metodo per risolvere il problema riscontrato nel precedente paragrafo è “stoppare” il task che percorre i quadrato quando si urta un oggetto.

Di seguito il codice NQC:

```
task check_sensors()
{
  while (true)
  {
    if (SENSOR_1 == 1)
    {
      stop move_square;
      OnRev(OUT_A+OUT_C); Wait(50);
      OnFwd(OUT_A); Wait(85);
      start move_square;
    }
  }
}
```

Questa é una buone soluzione. Per un piccolo task potrebbe essere l’ottimo ma spesso non è quello che vogliamo ottenere: noi desidereremo, una volta fatto ripartire il task, che questo continui da dove si era interrotto. Ciò non accade i quanto il comando di start (o l'equivalente C `execi()`), fa ripartire il task dalla prima istruzione, deleterio per task molto lunghi

10.3 Uso dei semafori

Una tecnica standard per risolvere questo problema è l’uso dei “*semafori*”, in modo che solamente un task alla volta possa accedere ai motori. Quando un task inizia ad utilizzare i motori pone il semaforo sul rosso, nel momento in cui un’altro task ne richiede l’utilizzo se il semaforo è verde ok, altrimenti aspetta finché il primo non la libera e pone il semaforo sul verde.

Dal punto di vista implementativo questo viene attuato mediante l’utilizzo di una costante che viene controllata ogni qual volta un task necessiti di utilizzare un motore, e settata come libera quando l’utilizzo è terminato

Di seguito il codice in NQC, la variabile del semaforo è `sem` ed il comando per generare l'attesa è :
`until(sem==0); sem=1`
Aspetta finché sem non diventa zero (semaforo verde) appena ciò accade lo pone ad 1 (semaforo rosso.
La trasposizione in legOS C è presto fatta utilizzando al posto dell’`until` la funzione :
`wait_event(&sem,0);`
Non vengono utilizzati i semafori forniti dal OS perché ad ogni check decrementano il valore del semaforo.

```
int sem;

task main()
{
  sem = 0;
  start move_square;
  SetSensor(SENSOR_1,SENSOR_TOUCH);
  while (true)
  {
    if (SENSOR_1 == 1)
    {
      until (sem == 0); sem = 1;
      OnRev(OUT_A+OUT_C); Wait(50);
      OnFwd(OUT_A); Wait(85);
      sem = 0;
    }
  }
}

task move_square()
{
  while (true)
  {
    until (sem == 0); sem = 1;
    OnFwd(OUT_A+OUT_C);           //OnFwd() in veritá sono 2 comandi
    sem = 0;                      // e non vanno interrotti
    Wait(100);
    until (sem == 0); sem = 1;
    OnRev(OUT_C);
    sem = 0;
    Wait(85);
  }
}
```

Tale metodo non è completo, per due task funziona ma cosa accade quando abbiamo diversi task che vogliono accedere ai motori? Chi ci indica l'ordine con cui è stato richiesto il suo utilizzo?.

10.4 Coda

Per risolvere il problema posto nel precedente paragrafo è necessario utilizzare un semaforo intelligente come un *coda* lunga quanti sono i task in cui viene scritto un codice identificativo diverso per ogni task.
Questo metodo è possibile solamente con il legOS, ricordando che "n" task (task2…taskn) vengono lanciati come:

```
pid2=execi(&task2, 0, NULL, 4, DEFAULT_STACK_SIZE);
pid3=execi(&task3, 0, NULL, 4, DEFAULT_STACK_SIZE);
...
pidn=execi(&taskn, 0, NULL, 4, DEFAULT_STACK_SIZE);
```

Basterà costruire una coda: una struttura di n puntatori in cui l'ultimo punta al primo. Quando un task richiede l'utilizzo di una risorsa inserirà nella prima locazione vuota il suo pid (pid2, pid3…pidn),ed il controllo avverrà sulla locazione alla sommità della coda, quando un task legge il suo pid significa che può utilizzare la risorsa, (il test si effettua sempre con `wait_event`). Da parte sua il task che utilizza la risorsa, una volta terminato, provvederà a rimuovere il suo pid dalla sommità della coda.

10.5 Time sharing

Vi è infine un altro metodo per il controllo di task paralleli, sensibile alla struttura fisica e meccanica dei sensori inseriti nel robot che interagiscono con l'ambiente.
Un esempio di questo si è visto nel Paragrafo 4.4.7 Follow Wall, se si va riosservare il codice si nota che quando si attiva il bump sensor, il semaforo non viene sbloccato una volta terminato l'utilizzo dei motori ma si fa girare il task per 3 cicli prima di rimettere il semaforo sul verde.
Questo accade perché una volta incontrato l'ostacolo, (la parete non convessa), il robot si gira fino a che il bump non è più sollecitato, se ora si rilascia l'uso dei motori, si riattiva il proximity sensor che fa dirigere l'RCX verso la parete, urto, si riattiva il bump… L'effetto macroscopico di tutto ciò è vedere il robot che compie piccole oscillazione su se stesso, in pratica bloccandosi.
Se si osserva più attentamente il tutto si nota come il robot alla prima abilitazione del bump , non si gira in direzione incidente alla parete minore di 90° , affinché ciò avvenga è opportuno lasciare ciclare per 3-4 volte il task del bump sensor da solo , in modo che l'RCX si giri di un po', continui il moto rettilineo, rivenga sollecitato il bump, si giri di un altro po', continui il moto rettilineo, rivenga sollecitato il bump e quest'ultima rotazione concluda il posizionamento con angolo minore di 90° verso al parete; ora si può riabilitare il task del proximity sensor.
Quindi, in questo metodo di coordinamento dei task, è stato individuato un tempo di attivazione minima per la corretta esecuzione del Behavior

10.6 Conclusione

Concludendo possiamo affermare che i metodi per la gestione dei task paralleli sono 2:

- Coda: Dipendente dal modo in cui il OS gestisce i task (software/hardware)
- Time sharing: Dipendente dall'interazione ambientale dei sensori per il compimento di un behavior (meccanica/fisica/ambiente)

Appendice 1 Pezzi lego

Tutti i pezzi lego disponibili nella scatola LEGO MINDSTORM ROBOTIC INVENTUION SYSTEM 1.5

Appendice 2 NQC quik refernce

Di seguito è riportata una lista di istruzioni, comandi e costanti usate dal NQC

2.1.1 Statements

Statement	Description
while (*cond*) *body*	Execute body zero or more times while condition is true
do *body* while (*cond*)	Execute body one or more times while condition is true
until (*cond*) *body*	Execute body zero or more times until condition is true
break	Break out from while/do/until body
continue	Skip to next iteration of while/do/until body
repeat (*expression*) *body*	Repeat body a specified number of times
if (*cond*) *stmt1* **if** (*cond*) *stmt1* **else** *stmt2*	Execute stmt1 if condition is true. Execute stmt2 (if present) if condition is false.
start *task_name*	Start the specified task
stop *task_name*	Stop the specified task
function(*args*)	Call a function using the supplied arguments
var = *expression*	Evaluate expression and assign to variable
var += *expression*	Evaluate expression and add to variable
var -= *expression*	Evaluate expression and subtract from variable
var *= *expression*	Evaluate expression and multiply into variable
var /= *expression*	Evaluate expression and divide into variable
var |= *expression*	Evaluate expression and perform bitwise OR into variable
var &= *expression*	Evaluate expression and perform bitwise AND into variable
return	Return from function to the caller
expression	Evaluate expression

2.1.2 Conditions

Sono usate per controllare gli statements e produrre decisioni. Molto spesso sono usate per confrontare expressions

Condition	Meaning
true	always true
false	always false
expr1 == expr2	test if expressions are equal
expr1 != expr2	test if expressions are not equal
expr1 < expr2	test if one expression is less than another
expr1 <= expr2	test if one expression is less than or equal to another
expr1 > expr2	test if one expression is greater than another
expr1 >= expr2	test if one expression is greater than or equal to another
! condition	logical negation of a condition
cond1 && cond2	logical AND of two conditions (true if and only if both conditions are true)
cond1 || cond2	logical OR of two conditions (true if and only if at least one of the conditions are true)

2.1.3 Expressions

Nelle espressioni possono essere usati diversi tipi di valori incluse costanti,variabili e sensor values. Da notare che SENSOR_1, SENSOR_2, e SENSOR_3 sono macro di SensorValue(0), SensorValue(1), e SensorValue(2) rispettivamente.

Value	Description
number	A constant value (e.g. "123")
variable	A named variable (e.g "x")
Timer(*n*)	Value of timer n, where n is between 0 and 3
Random(*n*)	Random number between 0 and n
SensorValue(*n*)	Current value of sensor n, where n is between 0 and 2
Watch()	Value of system watch
Message()	Value of last received IR message

Value possono essere combinati usando gli Operators , alcuni dei quali necessitano di operandi costanti, nel caso queste siano espressioni non devono involvere nulla ma essere costanti.
Di seguito la lista degli Operator in ordine di precedenza dal piú alto al piú basso

Operator	Description	Associativity	Restriction	Example
abs() **sign**()	Absolute value Sign of operand	n/a n/a		abs(x) sign(x)
++ --	Increment Decrement	left left	variables only variables only	x++ or ++x x-- or --x
- ~	Unary minus Bitwise negation (unary)	right right	constant only	-x ~123
* / %	Multiplication Division Modulo	left left left	constant only	x * y x / y 123 % 4
+ -	Addition Subtraction	left left		x + y x - y
<< >>	Left shift Right shift	left left	constant only constant only	123 << 4 123 >> 4
&	Bitwise AND	left		x & y
^	Bitwise XOR	left	constant only	123 ^ 4
|	Bitwise OR	left		x | y
&&	Logical AND	left	constant only	123 && 4
||	Logical OR	left	constant only	123 || 4

2.1.4 RCX Functions

Molte functions richiedono argomanti costanti. L'unica eccezione sono quelle che usano un *sensor* come argomento.

I nomi dei sensori sono: SENSOR_1, SENSOR_2, o SENSOR_3. In alcuni casi vi sono nomi predefiniti: (es. SENSOR_TOUCH) a cui vengono attribuiti valori costanti.

Function	Description	Example
SetSensor(*sensor, config*)	Configure a sensor.	SetSensor(SENSOR_1, SENSOR_TOUCH)
SetSensorMode(*sensor, mode*)	Set sensor's mode	SetSensor(SENSOR_2, SENSOR_MODE_PERCENT)
SetSensorType(*sensor, type*)	Set sensor's type	SetSensor(SENSOR_2, SENSOR_TYPE_LIGHT)
ClearSensor(*sensor*)	Clear a sensor's value	ClearSensor(SENSOR_3)
On(*outputs*)	Turn on one or more outputs	On(OUT_A + OUT_B)
Off(*outputs*)	Turn off one or more outputs	Off(OUT_C)
Float(*outputs*)	Let the outputs float	Float(OUT_B)
Fwd(*outputs*)	Set outputs to forward direction	Fwd(OUT_A)
Rev(*outputs*)	Set outputs to backwards direction	Rev(OUT_B)
Toggle(*outputs*)	Flip the direction of outputs	Toggle(OUT_C)
OnFwd(*outputs*)	Turn on in forward direction	OnFwd(OUT_A)
OnRev(*outputs*)	Turn on in reverse direction	OnRev(OUT_B)
OnFor(*outputs, time*)	Turn on for specified number of 100ths of a second. Time may be an expression.	OnFor(OUT_A, 200)

Appendice 3 Installare il legOS

3.1 LegOS 0.2.4 - Instructions for installing on NT/Win9x with Cygwin

---

[Masetti 00]These instruction works for me (I had NT WS 4.0 SP6a, now I'm using Win2k but some friends of mine use Win9x), I can't be sure they'll do for you but I'm quite sure. They may get legOS working, but they should be used at your own risk as with any other program installation, you may crash your computer, lose files, and generally make a mess of things. So, neither he nor I are responsible for what happens.

These instruction give you a fully functional LegOS 0.2.4, able to recompile the entire kernel image, the tools (dll, firmdl3 & so on). You will have to download a lot of MB. If you do not like that, please use WinlegOS that is a good porting of the minimal requirement to compile and upload .lx files with Windows.

3.1.1 Step by step instructions:

1) Install Cygwin32, the Unix development environment for Windows

- First, get the file cygwin.exe (which contains Cygwin). Remember the directory where this file is being saved to. Go to that directory and double click cygwin.exe to install cygwin32. Follow all of the default questions.
- Edit the path by: Go to Start-Run, and type: Sysedit. In the Autoexec.bat file add this to the "set path=" line: c:\cygnus\cygwin-b20\H-i586-cygwin32;c:\cygnus\cygwin-b20\H-i586-cygwin32\bin; then reboot your system.
- Note that this will only work (right now) with the older version of cygwin listed above. The cross-compiler was compiled against the libraries in this older version, and so unless someone creates a new cross-compiler linked to the new version of cygwin, an installation with that new version will inevitably fail.

If you don't have administrative access on an NT machine, you won't have to worry about the PATH thing until step 4.

2) Install the Hitachi-H8 cross-compiler

- Download the H8 compiler for windows from this site. Unzip the file you've just downloaded into the directory c:\cygnus\cygwin-b20\H-i586-cygwin32.

3) Install the LegOS 0.2.4 files

- Get LegOS-0.2.4.tar.gz. Drag a copy to C:\.
- Open a cygwin window (the bash shell that you can open with Start/Programs/Cygnus Solution/Cygnus B20) and type:

```
$ cd /
$ tar xvfz LegOS-0.2.4.tar.gz
```

- Lots of files will stream by as they install into their directories. This will create a legOS directory too.

Please, DO NOT USE any other program instead of tar (like WinZip or others) to decompress the original files: these program looses the symbolic links stored in the original tar file.

4) Edit Makefile.common

Remember, if you did not set your PATH in step 1, you'll have to set TOOLPREFIX to read "TOOLPREFIX=/cygnus/cygwin-b20/H-i586-cygwin32/bin/h8300-hms-". If you have set the path in step 1, you can skip to step 5.

- Open "/LegOS/Makefile.common" in Wordpad (or NoteTab, or VI if you like it) and edit as follows:

    a. There are two lines that start with TOOLPREFIX. The second one (which is preceded by the line "#NT") should look like "TOOLPREFIX=/cygnus/cygwin-b20/H-i586-cygwin32/bin/h8300-hms-".

- "File-Save" this file; if you get options to save as "rich text" or "Word format," make sure you save as "text only".

5) Install Perl

- Get ActivePerl from http://www.activestate.com/
- Install it.
- Execute the following commands to make a symbolic link for /usr/bin/perl so that cygwin can find ActivePerl:

```
$ cd /
$ mkdir usr
$ cd usr
$ mkdir bin
$ cd bin
$ ln -s /Perl/bin/perl perl
```

You need ActivePerl only for few scripts, but it's needed to get the full compile.

6) TRY IT!!!

- Type:

```
$ cd /legOS
$ make realclean
$ make depend
$ make
$ cd util
$ make strip
Now, Power on RCX.
Put IR tower on COM1
$ ./firmdl3 ../boot/legos.srec
$ ./dll ../demo/helloworld.lx
```

Press RUN on RCX and enjoy!

If you have IR tower connected to another com port you can change the default with the instruction: "SET DEFAULTTTY=COMx".

3.1.2 Final notes

If it doesn't work, consult lugnet.robotics.rcx.legos on lugnet.com.
Every command indicated are executed from Cygwin environment. In the examples, the "$" is the prompt cygnwin gives to you.
You can use Notetab Light to edit files instead of WordPad or VI. It's available from http://www.notetab.com/. It preserves the unix line-endings, and it can change between unix and dos line-endings.

Appendice 4 CVS legOS

4.1 Getting legOS Development Versions From CVS

4.1.1 Instructions

A number of folks have asked how to get legOS out of CVS, so here are the basic instructions. These are each separate commands, to be executed at the command prompt.

```
export CVS_RSH=ssh
export CVSROOT=:pserver:anonymous@cvs.legOS.sourceforge.net:/cvsroot/legOS
cvs login
cvs -z3 checkout legOS
```

Because CVS works poorly with symlinks, you'll have to add one link, using the following commands (adjust as necessary for your file system):

```
cd legOS/util/dll-src/
ln -s ../../kernel/lnp.c .
```

4.1.2 More Details on CVS at Sourceforge

If you want to know more about how to set up CVS with Sourceforge, you'll probably want to look at http://sfdocs.sourceforge.net/sfdocs/, which (about halfway down the page) provides a whole list of links about CVS, including basics, more detailed stuff, and CVS with Windows.

4.1.3 Getting CVS Write Access

Getting CVS write access really isn't that hard. The first thing to do is have a patch to add something useful to legOS. Preferably, the patch should apply cleanly against the current CVS version of legOS, and should be well commented. Markus did an excellent job of commenting his code, and new contributors should strive to do the same. Once you've done that, write to the legOS list at lugnet.com, describe your patch, and state that this is only the first in a long and woderful series of patches that will make legOS more stable and ensure world peace :) Additionally, you'll have to register as a developer at sourceforge. Once you've done that, we are normally pretty generous about giving write permission.

Appendice 5 LegOS command reference 0.2.4

[Chen 00]

5.1.1 Task Management

- void **tm_start();**
  Start Task Manager: tm.h, tm.c

- (int)pid **execi(*&PROCESS_NAME, int argc, char **argv,* priority_t priority, size_t stack_size)**
  Place function ***PROCESS_NAME*** into the Process queue, returns the Process's assigned ***PID***.: unistd.h, tm.c

- void **exit(int *code*);**
  Exits Process, returning ***code***.: unistd.h, tm.c

- void **kill(int *PID*);**
  Kill Process associated with ***PID*** as assigned when it was started by **execi()**: unistd.h, tm.c

- void **killall(priority_t *p*);**
  Kill all Processes with a Priority less than ***p***: unistd.h, tm.c

- wakeup_t **wait_event(wakeup_t(*wakeup) (wakeup_t), wakeup_t data);**
  Suspend current Process until Event wakeup function is non-null: unistd.h, tm.c

- void **yield();**
  Yield the rest of the current Task's timeslice.: unistd.h, tm.c

Predefined Priority Levels: P_DEAD, P_RUNNING, P_SLEEPING, P_WAITING, P_ZOMBIE, PRIO_HIGHEST, PRIO_LOWEST, PRIO_NORMAL : tm.h

5.1.2 Memory Management and String Operations

- void **free(void *the_ptr);**
  Free block of memory pointed to by ***the_ptr***.: stdlib.h, mm.c
- void ***calloc(size_t *nmemb*, size_t *size*);**
  Allocate adjacent blocks of memory, ***nmemb*** number of blocks of ***size*** individual block size.: stdlib.h, mm.c
- void ***malloc(size_t *size*);**
  Allocate a block of memory.: stdlib.h, mm.c
- void ***memcpy(void **dest*, const void **src*, size_t *size*);**
  dest = destionation address, ***src*** = source address, ***size*** number of bytes to copy : string.h
- void ***memset(void **s*, int *c*, size_t *n*);**
  Fill memory block at address ***s*** with byte value ***c***, ***n*** is the number of bytes of ***c*** to fill.
  string.h
- int **strcmp(const char **s1*, const char **s2*)**
  Compare two NULL terminated strings, returns: <0: s1<s2, = 0: s1==s2, >0: s1>s2 : string.h
- char ***strcpy(char **dest*, const char **src*);**
  Copy NULL-terminated string from ***src*** to ***dest***, returns pointer to ***dest***.: string.h

- int **strlen(const char *s)**
  Returns length of NULL-terminated string ***s*** : string.h
- NULL Null memory pointer constant.: mem.h

Semaphores
- sem_destroy() : semaphore.h
- sem_event_wait() : semaphore.c
- sem_getvalue() : semaphore.h
- sem_init() : semaphore.h
- sem_post() : semaphore.h, semaphore.c
- sem_t : semaphore.h
- sem_trywait() : semaphore.h, semaphore.c
- sem_wait() : semaphore.h, semaphore.c

5.1.3 Motor Control

- void **motor_a_dir(enum *MotorDir*)**
  void **motor_b_dir(enum *MotorDir*)**
  void **motor_c_dir(enum *MotorDir*)**
  Parameter: ***MotorDir*** Enumerated as: off = 0, fwd = 1, rev = 2, brake = 3
  Set the motor direction.: dmotor.h

- void **motor_a_speed(int *speed*)**
  void **motor_b_speed(int *speed*)**
  void **motor_c_speed(int *speed*)**
  Parameter: ***speed*** sets the PWM output to the specified motor.
  MAX_SPEED = Constant for upper limit of motor speed
  MIN_SPEED = Constant for lower limit of motor speed.: dmotor.h

5.1.4 RCX Button Input

- char **getchar()**
  Returns one of the Enumerated KEY types: KEY_ONOFF, KEY_PRGM, KEY_RUN, KEY_VIEW
  Input is debounced in dkey routines (unlike dbutton).: dkey.h, dkey.c

- event wakeup_t **dkey_pressed();** : dkey.h, dkey.c

- event wakeup_t **dkey_released();** : dkey.h, dkey.c

Enumerated KEY types: KEY_ANY, KEY_ONOFF, KEY_PRGM, KEY_RUN, KEY_VIEW.

5.1.5 Sensors

Defined Constants: LIGHT_RAW_BLACK = 0xffc0 (active light sensor raw black value), LIGHT_RAW_WHITE = 0x5080 (active light sensor raw white value). LIGHT_MAX = maximum decoded value at LIGHT_RAW_WHITE using the formula SCALED_LIGHT_READING = (147 - (RAW_LIGHT_READING >> 6) / 7).

- **DS_ALL_ACTIVE**
  Macro to set all Sensors ACTIVE: dsensor.c
- **DS_ALL_PASSIVE**
  Macro to set all Sensors PASSIVE: dsensor.c
- void **ds_active(*SENSOR*);**
  void **ds_passive(*SENSOR*);**
  Parameter: ***SENSOR*** = (&SENSOR_1, &SENSOR_2, &SENSOR_3) to active or passive type: dsensor.h
- void **ds_rotation_off(SENSOR);**
- void **ds_rotation_on(SENSOR);**
  Parameter: ***SENSOR*** = (&SENSOR_1, &SENSOR_2, &SENSOR_3) turns Rotation track off/on: dsensor.h
- void **ds_rotation_set(*SENSOR*, int *i*);**
  Sets Rotation ***SENSOR*** to arbitrary reading ***i*** : dsensor.h, dsensor.c
- boolean **TOUCH_1, TOUCH_2, TOUCH_3**
  Processed Touch Sensor reading: dsensor.h
- int **LIGHT_1, int LIGHT_2, int LIGHT_3**
  Scaled SENSOR reading.: dsensor.h
- int **ROTATION_1**, **ROTATION_2**, **ROTATION_3**
  Processed Rotation Sensor reading: dsensor.h
- int **SENSOR_1**, **SENSOR_2**, **SENSOR_3**
  Raw Sensor Input reading: dsensor.h
- int **get_battery_mv();**
  Get Battery level in XXXX mV: battery.h, battery.c
- int **BATTERY**
- Raw Battery Voltage level: dsensor.h

5.1.6 Display

Display positions
Digit display positions are denumerated from right to left, starting with 0 for the digit right to the running man. Digit 5 is only partially present on the RCXs display.

Native segment masks
In these bitmasks, bit 0 toggles the middle segment. Bit 1 toggles the top right segment, and the remaining segments are denumerated counterclockwise. The dot isn't encoded.

- void **cls ();**
  Clear user portion of screen.: conio.h, conio.c
- void **cputs (char * *s*);**
  Parameters: ***s*** the string
  Display an ASCIIZ string, only the first 5 characters will be displayed. if there are less than 5 characters, the remaining display positions will be cleared. : conio.h, conio.c
- void **cputw (unsigned *word*);**
  Parameters: ***word*** the hexword
  Display a hexword, position 0 not used.: conio.h, conio.c
- void **cputc_hex_X (unsigned nibble *c*);**
  void **cputc_hex (char *c*, int *X*);**
  Parameters: ***c*** hex number to display at position ***X***.:lcd.h
- void **lcd_digit(int *i*);**
  Parameter: ***i*** integer to display at position 0.: lcd.h

- void **lcd_clock(int *i*);**
  Parameter: ***i*** will be displayed with the format XX.XX
- void **lcd_number (int *i*, lcd_number_style *n*, lcd_comma_style *c*);**
  Number Style: digit, sign, unsign
  Comma Style: e0, e_1, e_2, e_3
  Parameters: ***i*** the integer to be shown, ***n*** the number style, ***c*** the comma style.: lcd.h
- void **lcd_hide(char *mask*);**
  void **lcd_show(char *mask*);**
  Parameters: ***mask*** see Mask listing below
  Display native mode segment mask at display position **X.**: lcd.h
  - LCD ***mask*** definitions
  - LCD_0_BOT, LCD_0_BOTL, LCD_0_BOTR, LCD_0_MID, LCD_0_TOP, LCD_0_TOPL, LCD_0_TOPR
  - LCD_1_BOT, LCD_1_BOTL, LCD_1_BOTR, LCD_1_MID, LCD_1_TOP, LCD_1_TOPL, LCD_1_TOPR
  - LCD_2_BOT, LCD_2_BOTL, LCD_2_BOTR, LCD_2_MID, LCD_2_TOP, LCD_2_TOPL, LCD_2_TOPR
  - LCD_3_BOT, LCD_3_BOTL, LCD_3_BOTR, LCD_3_MID, LCD_3_TOP, LCD_3_TOPL, LCD_3_TOPR
  - LCD_4_BOT, LCD_4_BOTL, LCD_4_BOTR, LCD_4_MID, LCD_4_TOP, LCD_4_TOPL, LCD_4_TOPR
  - LCD_5_MID
  - LCD_2_DOT, LCD_3_DOT, LCD_4_DOT
  - LCD_A_LEFT, LCD_A_RIGHT, LCD_A_SELECT
  - LCD_B_LEFT, LCD_B_RIGHT, LCD_B_SELECT
  - LCD_C_LEFT, LCD_C_RIGHT, LCD_C_SELECT
  - LCD_CIRCLE_0, LCD_CIRCLE_1, LCD_CIRCLE_2, LCD_CIRCLE_3
  - LCD_BATTERY_X
  - LCD_ARMS, LCD_1LEG, LCD_2LEGS, LCD_BODY
  - LCD_DOT_0, LCD_DOT_1, LCD_DOT_2, LCD_DOT_3, LCD_DOT_4,
  - LCD_IR_LOWER, LCD_IR_UPPER
  - LCD_EMPTY_1, LCD_EMPTY_2

5.1.7 LNP - LegOS Network Protocol

Don't have much information about this section yet, anyone who can shed some light on what LNP functions are present and how they work, please let me know.

- void **lnp_logical_fflush();**

Flush the Input Buffer: lnp-logical.h

- void **lnp_logical_range(int *i*);**

Parameter: ***i*** sets Range for IR: 0 = Short Range, 1 = Long Range
: lnp-logical.h

- int **lnp_logical_range_is_far()**

Returns the IR Range setting.: lnp-logical.h

- int **lnp_logical_write(const void* *buf*,size_t *len*);**

Parameter: ***buf*** string of length ***len*** is written to the IR Port: lnp-logical.h

5.1.8 Sound

- void **dsound_system(*SOUND*);**

- Pre Defined System ***SOUND***: DSOUND_BEEP : dsound.h

- Event **dsound_finished()**
Returns a Non-Zero if sound has finished playing.: dsound.h

- int **dsound_playing()**
Returns nonzero value if a sound is playing: dsound.h

- void **dsound_stop();**
Stop playing current sound/song.: dsound.h

- void **dsound_play(const note_t **notes*);**
Parameter: ***notes*** array of note_t as defined below:
Note Data Type:
typedef struct {
  unsigned char pitch; //!< note pitch, 0 ^= A_0 (~55 Hz)
  unsigned char length; //!< note length in 1/16ths
} note_t;: dsound.h

    - Pre Defined Note Lengths: WHOLE, HALF, QUARTER, EIGHTH

    - Pre Defined Pitches (Octave **X** = 0-7):
PITCH_A**X**, PITCH_Am**X**, PITCH_C**X,** PITCH_Cm**X,**
PITCH_D**X,** PITCH_Dm**X,** PITCH_E**X,** PITCH_F**X,**
PITCH_Fm**X,** PITCH_G**X,** PITCH_Gm**X,** PITCH_H**X,**
PITCH_END, PITCH_MAX, PITCH_PAUSE : dsound.h

5.1.9 Miscellaneous

- void **main();**
Main entry point into the User Program.: kmain.c
- int **sleep(int *sec*);**
Pause for ***sec*** seconds ...
int **msleep(int *msec*);**
Pause for ***msec*** milliseconds before executing next commands in current task, other tasks will continue to execute commands uneffected.
(NOT IMPLEMENTED: Returns 0 if interrupted, otherwise returns number of msec until interrupted): unistd.h, tm.c
- void **power_off();**
Initiates Software Standby/Low Power mode. On/Off button will reactivate.: system.h
- void **program_run(unsigned *P*);**
Execute Program in slot number ***P.***: program.c
- void **reset();**
Returns control to ROM, ie. Cold Boot.: system.h

- int **random()**
  Returns a random integer.: stdlib.h
- void **srandom(int *seed*);**
- Seeds the Random Number Generator.: stdlib.h
- long int **sys_time**
  Current System Time (Time up from last firmware d/l and boot up) in msec.
  This is a 32 bit value which will overflow after 49.7 days of continuous operation:
  time.h, systime.c

.

Appendice 6 LNP command example

The command-line options for running lnpd are explained in the README, but there are a couple that are particularly useful. One is the **--nolock** option. Normally, lnpd tries to lock the serial port on startup, but permissions to do that are usually denied for users. Lnpd will exit if it fails to create a lock. The **--nolock** option will keep lnpd from even trying, and so it will run successfully. Second is the **--log[=filename]** option. This option makes lnpd log messages to the specified file, or to the system log if no file is specified. The log file can be extremely useful for tracking down problems. If your IR tower isn't on the port lnpd expects, you may specify another port with **--tty=<device>**. So, putting all this together, when I start up lnpd my command line looks like this:

```
./lnpd --nolock --log=foo (My tower is on the default port.)
```

The daemon starts up, and the command prompt returns. Assuming all goes well, the green LED on your IR tower should be lit (it will go off in a few seconds), and you should be able to use LNP programs. However, any other program that uses the serial port (firmdl3, normal dll) will not work. This is why the version of dll that comes with the LNP package is so valuable, as it allows you to dowload new versions of your program without killing lnpd every time.

6.1.1 Headers

As always, you have to include headers in order to use LNP. On the RCX, the #include line is **#include <lnp.h>** and on the PC it is **#include "liblnp.h"**.

6.1.2 Initialization

On the PC, LNP must be initialized before use. LNP is always running on the RCX, but it is still a good idea to do a little setup before using it.

On the PC, the function to call is **lnp_init(tcp_hostname, tcp_port, lnp_address, lnp_mask, flags)**. For all parameters, 0 means default. PC programs connect to lnpd over TCP, so **tcp_hostname** and **tcp_port** are the hosntame and port of the lnpd daemon to connect to. (Yes, you can actually have an LNP-using program running on one computer and have the daemon and IR tower on another.) **lnp_address** is the network address for your program on the LNP network. **lnp_mask** determines which bits of the eight-bit LNP address determine the network device, and which bits determine the port number. **flags** can change the behavior of lnpd in certain situations. There is rarely a reason to change the defaults for this call. **lnp_init()** will return 0 if init was successful, non-zero if it was not successful.

```
if(lnp_init(0,0,0,0,0))
{
  perror("lnp_init");
  exit(1);
}
else
  printf(stderr,"init OK\n");
```

It's good to have a set of #defines with your ports, the remote host, and its ports. In my RCX program, these look like this:

```
#define MY_PORT 2
```

```
#define DEST_HOST 0x8
#define DEST_PORT 0x7
#define DEST_PORT_2 0x8
#define DEST_ADDR ( DEST_HOST << 4 | DEST_PORT )
#define DEST_ADDR_2 ( DEST_HOST << 4 | DEST_PORT_2 )
```

In other words, my program's port is port 2. The remote host is at address 8, and I'll be sending messages to ports 7 and 8 on the remote host. With the default mask, the upper four bits of the address determine the host, and the lower four bits determine the port on that host. When transmitting data, the address must be passed to LNP with the host and port already stuffed into one byte, which is why I have those last two #defines.

6.1.3 Packet Handlers

In order to recieve data with your program, you must set up packet handlers. A handler will be called whenever a packet arrives on that handler's port. You must set up a handler for each port you expect to recieve data on. Addressing handlers are defined as follows:

```
void addr_handler_1(const unsigned char *data, unsigned char length, unsigned
char src);

lnp_addressing_set_handler (MY_PORT_1, addr_handler_1);
```

6.1.4 Sending Data

Sending data is pretty simple as well. You need to have some data to send, of course. You need to know how big the data is. Finally, you need an address to send it to.

```
result = lnp_addressing_write(data, length, DEST_ADDR, MY_PORT);
```

6.1.5 Debugging

```
void printString(char *s)
{
  int len, result;
  char *buffer;

  len = strlen(s);
  buffer = malloc(len + 2);
  buffer[0] = 's';
  memcpy(buffer + 1, s, len + 1);
  result = lnp_addressing_write(buffer,len + 2, DEST_ADDR,   MY_PORT);
  free(buffer);
}
```

This function takes a string as input, and sends it out with a type identifier of 's' (for string) in front of it. First we need to know how long the string is, then a temporary buffer is allocated. All of the data to be sent has to be in one buffer, so some scratch space is needed to build the packet in. That scratch space is allocated, the identifier is placed into the first byte, and the string (including terminating null) is copied in using memcpy. Finally, the whole thing is sent out with **lnp_addressing_write()**, and the buffer is deallocated.

Note that an array for a buffer would be faster than allocating and deallocating a buffer for each function call. There are disadvantages, however. If the array is a normal stack variable, then it will take up a large amount of stack space, which could cause a problem with overflows. If the array is declared static, then you must ensure that only one thread at a time calls the function, either by design or with semaphores.

```
void printInt(int i)
{
  char buf[3];
  int result;

  num = (char *)(&i);
  buf[0] = 'i';
  memcpy(buf + 1, &i, 2)
  result = lnp_addressing_write(buf,3, DEST_ADDR, MY_PORT);
}
```

This function takes a single int and sends it over the network. Here the buffer is just a local array, since it only needs to be three bytes long. The first byte of the buffer is set to the type identifier ('i', for int), and the following two are set to the two bytes composing the int. Finally, again, the buffer is sent over the network with **lnp_addressing_write**.

Now we have the LNP handler on the PC side:

```
void addr_handler(const unsigned char* data,unsigned char length, unsigned char
src)
{
  short temp;
  char *ptr;

  switch(data[0])
  {
  case 's':
    puts(data + 1);
    break;
  case 'i':
    ptr = (char *)□temp;
    #ifdef BIG_ENDIAN
      ptr[0] = data[1];
      ptr[1] = data[2];
    #else
      ptr[0] = data[2];
      ptr[1] = data[1];
    #endif
    printf("%d", temp);
    break;
  }
}
```

The basic idea here is to examine the type identifier in the packet and then either print the string that follows or extract the int and print it. Some trouble comes when considering so-called endian issues. (If you know what endian is and how to deal with it, skip to the paragraph after next.) The short version is, there are two main ways to represent numbers that take up multiple bytes in memory. "Big-endian" processors store the most-significant byte at the lowest address, and the least-significant byte at the highest address. If the number 0x1234 is stored at location 100, then the byte at 100 contains 0x12 and the byte at 101 contains 0x34. Little-endian processors store things in the opposite direction, so that byte 100 would contain 0x34 and byte 101 contains 0x12.

The good news is, you almost never have to deal with this. As long as your processor is consistent (and if it wasn't, it wouldn't work) and you don't do anything evil like examining the individual bytes of a number, you'll never notice it. However, there is a problem when it comes to transferring data from one computer to another. If the two computers have different byte orders, any numbers transferred will appear scrambled. To fix this, the bytes must be swapped, and that's why the indices into **data** are backwards in the **#else** clause in the function. The RCX's processor is big-endian, so if the host PC is also big-endian then the two processors can use each others data without modification. Of course, the **BIG_ENDIAN** identifier is fictional: you have to figure out what your computer is for yourself. It's not hard. If you're running an Intel-based PC, your processor is little-endian. If you're running a PowerPC, it's big-endian. If you're running something else, chances are it's big-endian, but you can also use the program below to check:

```
int main(void)
{
  int x = 1;
  char *foo = (char *)&x;
  if(foo[0] == 1)
    printf("little-endian\n");
  else
    printf("big-endian\n");
  return 0;
}
```

Now you have a much better debugging system! These function calls can be called pretty much anywhere, anytime, with the exception of LNP data handlers. There are a couple of things to watch out for, though. One problem is that these functions take some time to run, since they don't return until the full packet has been transmitted. Keep that in mind when debugging time-critical routines. Also, these strings are actually stored on the RCX, and sent over the network every time they are printed. This takes up limited memory and time. In my project, turning off debugging strings reduces my program size by over 1k, a significant portion of the program. If memory is a problem, it may be worth the trouble to send string ID numbers from the RCX which the PC would then look up in a table to get the actual string.

Once you understand the principles above, it should be pretty easy to move past simple debugging and into real data exchange. Keep in mind endian issues when transmitting any numbers bigger than one byte if your machine is little-endian.

6.1.6 Remote Control

Sending data from the RCX to the PC is pretty simple, since anything goes in an LNP packet handler on the PC. On the RCX side, however, an LNP packet handler is limited in what it can do. Things like memory allocation and thread controls are off-limits. Heavy data processing is highly discouraged. In general, it's best to simply copy the data into a buffer, set a flag, and let a thread take care of the processing. An example setup of this follows:

```
int gNewData = 0;
int gMessagingData[256];
int gDataLength;

void packet_handler(const unsigned char* data,unsigned char length, unsigned
char src)
{
  if(gNewData == 0)
  {
    memcpy(gMessagingData, data, length);
```

```
    gDataLength = length;
    gNewData = 1;
  }
}

int PacketWatcher(int argc, char **argv)
{
  wakeup_t WaitForData(wakeup_t data)
  {
    return gNewData;
  }

  while(1)
  {
    wait_event(WaitForData, 0);
    switch(gMessagingData[0])
    {
      ...
    }
    gNewData = 0;
  }
}
```

All the packet handler does is copy the data to gMessagingData and then set gNewData, but only if gNewData is clear. That flag is the signal for the packet watcher thread to look at the new data. If any packets comes in before the packet watcher can take care of the data, those packets will be discarded. If that is a problem, it may be possible to use the pool more efficiently, or use some other scheme. Since the legOS scheduler does not support explicitly waking threads, the packet watcher has to wait for its turn to be touched before it can go into action. Again, the first byte of the packet is used as an identifier to figure out what to do with the rest of the data. Fill in the blank in the switch statement to do whatever you need. Finally the flag is reset, signalling that the packet watcher is ready to process more data.

On the PC side, send data just like in the RCX programs. Keep in mind byte-order issues if your machine is little-endian. Again, since your PC has the processing power and the memory, I recommend doing the byte swapping in the PC program, rather than on the RCX.

Appendice 7 LegOS Semaphore

SEMAPHORES(3) SEMAPHORES(3)

NAME

sem_init, sem_wait, sem_trywait, sem_post, sem_getvalue,sem_destroy - operations on semaphores

SYNOPSIS

```
#include <semaphore.h>

int sem_init(sem_t *sem, int pshared, unsigned int value);

int sem_wait(sem_t * sem);

int sem_trywait(sem_t * sem);

int sem_post(sem_t * sem);

int sem_getvalue(sem_t * sem, int * sval);

int sem_destroy(sem_t * sem);
```

DESCRIPTION

This manual page documents POSIX 1003.1b semaphores, not to be confused with SystemV semaphores as described in
ipc(5), semctl(2) and semop(2).
Semaphores are counters for resources shared between threads. The basic operations on semaphores are: increment the counter atomically, and wait until the counter is non-null and decrement it atomically.

sem_init initializes the semaphore object pointed to be sem. The count associated with the semaphore is set initially to value. The pshared argument indicates whether the semaphore is local to the current process (pshared is zero) or is to be shared between several processes (pshared is not zero). LinuxThreads currently does not support process-shared semaphores, thus sem_init always returns with error ENOSYS if pshared is not zero.

sem_wait suspends the calling thread until the semaphore pointed to by sem has non-zero count. It then atomically decreases the semaphore count.

sem_trywait is a non-blocking variant of sem_wait. If the semaphore pointed to by sem has non-zero count, the count is atomically decreased and sem_trywait immediately returns 0. If the semaphore count is zero, sem_trywait immediately returns with error EAGAIN

sem_post atomically increases the count of the semaphore pointed to by sem. This function never blocks and can safely be used in asynchronous signal handlers.

sem_getvalue stores in the location pointed to by sval the current count of the semaphore sem.

sem_destroy destroys a semaphore object, freeing the resources it might hold. No threads should be waiting on the semaphore at the time sem_destroy is called. In the LinuxThreads implementation, no resources are associated with semaphore objects, thus sem_destroy actually does nothing except checking that nothread is waiting on the semaphore.

CANCELLATION

sem_wait is a cancellation point.

ASYNC-SIGNAL SAFETY

On processors supporting atomic compare-and-swap (Intel 486, Pentium and later, Alpha, PowerPC ,MIPS II, Motorola 68k), the sem_post function is async-signal safe and can therefore be called from signal handlers. This is the only thread synchronization function provided by POSIX threads that is async-signal safe.
On the Intel 386 and the Sparc, the current LinuxThreads implementation of sem_post is not async-signal safe by lack of the required atomic operations.

RETURN VALUE

The sem_wait and sem_getvalue functions always return 0.
All other semaphore functions return 0 on success and -1 on error, in addition to writing an error code in errno.

ERRORS

The sem_init function sets errno to the following codes on error:

EINVAL value exceeds the maximal counter value SEM_VALUE_MAX

ENOSYS pshared is not zero

The sem_trywait function sets errno to the following error code on error:

EAGAIN the semaphore count is currently 0

The sem_post function sets errno to the following error code on error:

ERANGE after incrementation, the semaphore value would exceed SEM_VALUE_MAX (the semaphor count is left unchanged in this case)

The sem_destroy function sets errno to the following error code on error:

EBUSY some threads are currently blocked waiting on the semaphore.

AUTHOR

Xavier Leroy <Xavier.Leroy@inria.fr>

SEE ALSO

pthread_mutex_init(3), pthread_cond_init(3), pthread_cancel(3), ipc(5).

Appendice 8 Esempi di programmazione

Di seguito vengono riportati alcuni esempi di programmazione.
Qui sotto viene riportata la licenza MPL una volta sola per tutti i file

```
/*-------------------------------------------------------------------------*\
 | The contents of this file are subject to the Mozilla Public License
 |  Version 1.0 (the "License"); you may not use this file except in
 |  compliance with the License. You may obtain a copy of the License at       |
 |  http://www.mozilla.org/MPL/
 |
 |  Software distributed under the License is distributed on an "AS IS”
 |  basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the
 |  License for the specific language governing rights and limitations
 |  under the License.
 |
 |  The Original Code is legOS code, released November , 2000.
 |  The Initial Developer of the Original Code is Maragno Simone.
 |  Portions created by Maragno Simone are Copyright (C) 2000
 |
 |  Maragno Simone. All Rights Reserved.
 |  Contributor(s): Maragno Simone <simomail@inwind.it>
                                                  |
 \*-------------------------------------------------------------------------*/
```

8.1.1 File Include_sensor

Esempi di utilizzo dei sensori

```
/*#include <conio.h>
#include <unistd.h>

#include <dsensor.h>
#include <sys/irq.h>
#include <dlcd.h>

#include <sys/battery.h>
#include <dmotor.h>
*/


  //cputs("light");  /*indicate which function we are in*/

  //ds_active(&SENSOR_2);    /* turn on light sensor LED (active mode)*/
  //lcd_int(LIGHT_2);
  // lcd_int(ds_scale(SENSOR_2));          //ritorna num 0-1024  (ov-->1023)
  // lcd_int(SENSOR_2);            // torna -9999 --> troppo grande per stare
in un int
  //lcd_int(ds_scale(BATTERY));          // torna 0-1024 (271)

// Restituisce valolre sensor_2  in mv    approssimando l'ultima cifra
// Per fare una giusta approssimax. :
// staccando sensor scal=1023 -> 4.88 mv , b prima della modifica= 4 , b dopo=5
/*int mv2(){
  int scal=ds_scale(SENSOR_2);
  long bh=( scal*0xC350L/0x3ffL );  // (*50000)
  long b=(scal*0x1388L/0x3ffL );   // (* 5000)
  b=((bh-b*10)>5)? b+1:b;
  return(int)( b);
 }
```

```
// Senor_2  conversione in millivolt
int mv2(){
  int scal=ds_scale(SENSOR_2);
  long bh=( scal*0xC350L/0x3ffL );  // (*50000)
  long b=(scal*0x1388L/0x3ffL );   // (* 5000)
  b=((bh-b*10)>5)? b+1:b;
  return(int)( b);
 }

//altro metodo per l'approssimazione ;
//+0x200  serve per approssimare il decimo bit quello che  poi diverrá il meno
significativo.
int mv2(){
  int scal=ds_scale(SENSOR_2);
  long b=(((scal*0x1388L)+0x200)/0x3ffL );   // (* 5000)
  return(int)( b);
 }
```

8.1.2 Uso dei tasti

Questo estratto di programma permette attraverso l'uitizzo dei tasti dei poter scegliere se visualizzare il valore del sensore sulla ruota destra o sinistra, e con un altro di visualizzare il valore incrementale ROTATION_1 e ROTATIN_3 , oppure il valore in volt

```
/*! \file   odo2.c
    \brief  gestione task  attraverso i bottoni
            creazione e modifica delle funzioni count_l_h e count_r_
    \author Maragno Simone <simomail@inwind.it>
*/

#include <conio.h>
#include <unistd.h>
#include <dsensor.h>
#include <semaphore.h>
#include <time.h>
#include <dmotor.h>
#include <dbutton.h>
#include <dsound.h>
#include <rom/system.h>
#include <../programmi/include-sensor.c>
#include <../programmi/odometry.c>
#define mv1 (mv1())
#define mv3 (mv3())

void change_sensor();
void change_task();
void tensioni();
void test();
void tm_start();
static int s=1,t=1;
pid_t pid1,pid4,pid5,pid6,pid7;


//   button press functions
//
static wakeup_t button_press_wakeup(wakeup_t data) {
      return PRESSED(dbutton(),data);
}
static wakeup_t button_release_wakeup(wakeup_t data) {
```

```
      return RELEASED(dbutton(),data);
}
static wakeup_t wait_clear_count_h (wakeup_t data) {
      return ((COUNT_STATE)&&(data));
}

//
//Quando schiacciamo button View la variabile s passa da -1 ad 1 e vice versa
// Serve per cambiare la visualizzazione tra sensore ruota destra  e sinistra
//
void change_sensor(){
  while(1){
      wait_event(&button_press_wakeup,BUTTON_VIEW); //wait for VIEW button
    wait_event(&button_release_wakeup,BUTTON_VIEW);//wait for release
    //cputw(dbutton());
    s*=-1;
  }
}

//
//ad ogni press del button Program vengono attivati alcuni task e terminati al-
tri.
//
void change_task(){
  while(1){
      wait_event(&button_press_wakeup,BUTTON_PROGRAM); //wait for PROGRAM button
      wait_event(&button_release_wakeup,BUTTON_PROGRAM);//wait for release
      t=0;
      pid7=execi(&test, 0,NULL, 4, DEFAULT_STACK_SIZE);
      wait_event(&button_press_wakeup,BUTTON_PROGRAM); //wait for PROGRAM button
      wait_event(&button_release_wakeup,BUTTON_PROGRAM);//wait for release
      kill(pid7);
      motor_c_dir(off);
      motor_b_dir(off);
      pid5=execi(&tensioni, 0, NULL, 4, DEFAULT_STACK_SIZE);
      wait_event(&button_press_wakeup,BUTTON_PROGRAM); //wait for PROGRAM button
      wait_event(&button_release_wakeup,BUTTON_PROGRAM);//wait for release
      kill(pid5);
      }
}

//
//Visualizza il valore in millivolt rilevato dai sensori 1 e 3
//
void tensioni(){
      ds_active(&SENSOR_1);
      ds_active(&SENSOR_3);
            while(1){
            //lcd_int(get_battery_mv());
            (s>0)?lcd_int(mv3):lcd_int(mv1); //Quando schiaccio View visualizza
l'uno o l'altro
            msleep(100);
            /*lcd_int(mv1());
            //Blocco per la visualizzazione in sequenza , compresa la tensione
delle pile
            sleep(1);
            lcd_int(mv2());
            sleep(1);
            lcd_int(mv3());
            sleep(1);
            lcd_int(get_battery_mv());
            sleep(1);*/
      }
```

```
}

void test(){
  int speed=200,error;
  rotation_1_set(odometry,0);
  rotation_3_set(odometry,0);
  rotation_1_on;
  rotation_3_on;
  motor_a_dir(fwd);
  motor_c_dir(fwd);
  motor_a_speed(speed);
      motor_c_speed(speed);
      while(ROTATION_1 <= 108){
      //lcd_int(ROTATION_3 );
      //msleep(100);
  }
  pid1=execi(&xy, 0,NULL, 4, DEFAULT_STACK_SIZE); //viene calcolata la posizione
  motor_a_dir(rev);                               //funzione esterna
  motor_c_dir(rev);
  //while(ROTATION_1 != 0);
  while(ROTATION_1 > -72);   //2 giri indietro
  error=ROTATION_1-ROTATION_3;
  motor_a_dir(off);
  motor_c_dir(off);
  dsound_system(DSOUND_BEEP);
  lcd_int(ROTATION_3);
  msleep(500);
  lcd_int(ROTATION_1);
  msleep(500);
  lcd_int(error);
}

int main(int argc, char *argv[]) {
   ds_active(&SENSOR_1);
   ds_active(&SENSOR_3);
   //ds_rotation_set(&SENSOR_1,odometry);
   //ds_rotation_on(&SENSOR_1);
   //pid5=execi(&tensioni, 0, NULL, 2, DEFAULT_STACK_SIZE);
   pid4=execi(&change_sensor, 0, NULL, 4, DEFAULT_STACK_SIZE);
   pid6=execi(&change_task, 0,NULL, 4, DEFAULT_STACK_SIZE);
   //pid7=execi(&test, 0,NULL, 2, DEFAULT_STACK_SIZE);
}
```

BIBLIOGRAFIA

Angeli Frank, 1999. Personal comunication to Gasperi M.. frank@frankweb.com

Angeli Frank, 1998. "Sensor light detection ", in *Mail to lego_robotic@crynwr.com*

Arbib, Kfoury, Moll, 1981, *A Basis of Theoretical Computer Science*, Spinger-Verlag, New York

Arbib M. A. 1992,"Schema Theory", in *The Enyclopedia Of Artificial Intelligent,* 2nd ed.,ed. S. Shapiro,Wiley-Interscience, New York, N.Y.

Arkin Ronald C, **1989a**. "Neuroscence In Motion: The Application Of Schema Theory To Mobile Robotics"in *Visuomotor Coordination:Amphibias, Comparasion, Models,And Robotics*, ed J.P.Ewert and Arbit, New York:Plenum Perss

Arkin Ronald C, **1989b** "Motor Schema-Based Mobile Roboti Navigation", in *International Journal Of Robotics Research,* Vol 8 No. 4

Arkin Ronald C, **1998**. "Behavior-Based Robotics", The MIT Press

Ash Mike, 2000. Curatore della sezione relativa al LNP nel LegOS Howto di Luis Villa mail@mikeassh.com

Baum's Dave, **2000**."Definitive Guide to Lego Mindstorms". book

Ben-Ari M., 1982. "Principles of Concurrent Programming", *Prentice-Hall International Inc*. 1982

Borenstein J, and Koren Y, 1985. "A mobile Platform For Nursing Robots" *IEEE Transactions on Idustrial Electronics*, Vol 32, No. 2,

Borenstein J, 1994a, "The CLAPPER: A Dual-drive Mobile Robot with Internal Correction of Dead-reckoning Errors", *Proceedings of IEEE Inrternational conference on Robotics and Automation*, San Diego, CA, 8-13 maggio

Borenstein J and Feng L., 1995a „Measurement and Correction of Systematic Odometry Errors in Mobile Robots" in *IEEE Transactions on Robotics and Automation,* aprile

Borenstein J and Feng L., 1995 b, "Correction of Systematic Dead-reckoning Errors in Mobile Robots" *Proceedings of the 1995 International conference on Intelligent Robots and Systems (IROS'95),* Pittsburgh, PA, 5-9 agosto

Borenstein J, Everett H.R. e Feng L., **1996** ."Where am I? Sensors and Methods for Mobile Robot Positioning", Univertity of Michigan, aprile

Brooks A. Rodney, 1991."Intelligene without reason", in *Proceding of the IJCAI,* Los Angeles: Morgan Kaufmann Pub.

Brooks R. 1990a. "The Behavior Language" *A.I. Memo No. 1227,* MIT AI Laboratory, aprile

Brooks R., 1986. "A Robust Layered Control System For A Mobile Robot", in *IEEE Journal of Robotics and Automation,* Vol. RA-2, No. 1

Byrne R.H., Klarer P.R. and Pletta J.B., 1992. "Techniques for Autonomous Navigation" in *Sandia Report SAND92-0457,* Sandia National Laboratories, Albuquerque, NM, marzo

Chen C.David, 2000,"legOS command reference" mailto:dcchen@pacbell.net

Chenavier F. and Crowley J., 1992. "Position Estimation for a Mobile Robot Using Vision and Odometry" *Proceedings of IEEE International Conference on Robotics and Automation*, Nice, France, 12-14 maggio

Clemente G, Moro M. 1999."Sistemi Operativi", edizioni Progetto , Padova

Cox I.J., 1991 "Blanche – An Experiment in Guidance and Navigation of an Autonomous Mobile Robot" *Transactions Robotics and Automation*, Scottsdale, AZ 14-19 maggio

Christiansen M.K., Pedersen M.H. and Glæsner, 2000. "Solving the priority inversion problem in legOS. Technical Report", *AUC*

Christiansen M.K., Pedersen M.H. and Rasmussen E.B., 2000. "Priortized interrupts in legOS. Technical Report", *AUC*

Crowley J.L. and Reignier P. 1992 "Asynchronous Control of Rotation and Translation for a Robot Vehicle" *The International Journal of Robotics Research,* Vol. 13 No. 3, giugno.

Clark Dennis, 2000. "Robotics projects", in http://www.verinet.com/~dlc/ projects/botproj.htm

Evans J.M., 1994, "HelpMate: An Autonomous Mobile Robot Courier for Hospitals", *1994 International Conference on Intelligent Robots and Systems (IROS'94),* Munich, Germany, 12-16 settembre

Everett, H.R., 1995 "Sensors for Mobile Robots: Theory and Application", *ISBN 1-56881-048-2,* A K Peters, Ltd, Wellesley, MA

Fogel Karl, 2000. Part of book "Open Source Development Whit CVS" in http:// cvsbook.red-bean.com/cvsbook.html#Basic_Concept

Gasperi Michel, 1998. "Mindstorm RCX Sensor Input Page gasperi@alink.com

Ghollingum J., 1990 "Caterpillar make the earth move: automatically", *The Industrial Robot*, Vol 18, No. 2

Hitachi , Hitachi Single-Chip Microcomputer H8/3297 Series hardware manual

http 1, http://news.lynet.com/robotics/RCX/legOS/?n=788

http 2, http://LeOS.sougeforge.net

http3, http://www.mozilla.org/MPL Mozilla Public License

Jakobi Nik, **1998**."Minimal Simulation for Evolutionary Robotics", University of Sussex, maggio

Jones Joseph L, 1993. "Mobile Robots. Inspiration to implementation", Peters

Kaelbling & Rosenschein, 1991. "Action and Planning in Embedded Agents", in Idesigning Autonomous Agents, ed P. Maes, MIT Press, Cambridge, MA

Kelso Scott J.A.G, 1995 "Self Organisation of Brain and Behavior", in *Dynamic Pattern: the Self-Organisation of brain and behavior*, MIT Press

Klarer P.R., 1988 "Simple 2-D Navigation for Wheeled Vehicles", *Sandia Report SAND88-0540,* Sandia National Laboratories, Alburquerque, NM, aprile

Lerman Kristina, "Design and Mathematical Analysis of Agent-based Systems"

Linkeman Steve,1996. in www.net.net/~stevlim/

Masetti Paolo, 2000. in http// paolom@pragmanet.it Giugno

Mataric Maja J. e **Monica Nicolescu**, **2000**" Extending Behavior-Based Systems Capabilities Usion An Abstract Behavior Representation", in the Working Notes of the *AAAI Fall Sumposium on Parallel Cognition*, North Fallmouth, 3-5 novembre

Mataric Maja J., **2000a.**" Capter 8 Robot control, Capter 7 Sensor" Capitoli in linea

Mataric Maja J., **2000b.**"CSCI445 Course note" in *http://www.robotic.usc.edu/~maja/* **,** Computer Scienze Department University of Southern California.

Mataric Maja J., **2000c**."Multiple Objective vs. Fuzzy Behavior Coordination", in *Fuzzy Logic Techniques for Autonomous Vehicle Navigation*, D. Drainkov e A. Saffiotti, eds. Studies on Fuzziness and Soft Computing, Springler-Verlag,

Mataric Maja J., **2000d**." Robust Behavior-Based Control for Distributed Multi-Robot Collection Tasks",*USC Institute for Robotics and Intelligent System Technical Report* IRIS-00-387, giugno

Mataric Maja J., **1999**." A Decision-theoretic approach to fuzzy behavior coordination", in *Proceedings, IEEE International Symposium on Computational Intellig ence in Robotics & Automation (CIRA-99)*, Monteray, 8-9 novembre

Mataric Maja J., **1998**." Learning from History for Adaptive Mobile Robot Control", in *Proceedings, IROS-98,* Victoria, Canada, 12-16 ottobre

Mataric Maja J., **1997**." Behavior-Based Control: Examples fron Navigation, Learning, and Group Behavior", *Journal of Experimental and Theoretical Artificial Intelligence*, special issue on Software Architectures for Psysical Agents, 9 (2-3), H. Hexmoor, I. Horswill, e D. Kortemkamp

Mataric Maja J., **1992**." Behavior-Based Control: Main Properties and Implications", in *Proceedings*, *IEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems*, Nizza, maggio

Mataric Maja J.,"On-line Modeling of Robot Interaction Dynamic"

Mataric Maja J., **http1**"Analog Sensor Problem", in *Cap 8: Robot contol, in rete*

McFarland David, 1992."Autonomy and self_sufficiency in robots", VUB AI Lab. Brussels

Mian Gian Antonio, 1999. "Appunti Di Elaborazione Numerica Dei Segnali", Edizioni Libreria progetto, Padova

Myler Harley R., **1999** ."Fundamentals of Machine Vision", SPIE Optical Engineering Press

Noga Markus, 98,"official legOS homepage", http://ww.noga.de/LegOS

Oboe Roberto,**1998** " Ingegneria e tecnologia dei sistemi di controllo", in *Appunti dalle lezioni, CUSL*

Oppenheim Alan V., Shaker Ronald, 1999."Discrete-Time Signal Processing", in Prenceton Hall Signal Processing Series, Alan V**.** Oppenheim Editor
Osborn Chris, 1998. Personal communication to Proudfott Keoka

Parker E. Lynne, **Bekey George** e **Barhen Jacob, 2000**. "Current State of the Art in Distributed Autonomous Mobile Robotics", in *Distributed Autonomous Robotic System 4*, Springer

Parker E. Lynne, **2000**. "Evaluating Success in Autonomous Multi-Robot Teams: Experiences from ALLIANCE Architecture Implementations", in *Journal of Theoretical and Experimental Artificial Intelligence*, special issue on *Autonomy Control Software*

Parker E. Lynne, **1998**."Distributed Control of Multi-Robot Teams: Cooperative Baton Passing Task", in *Proceedings on the 4th Intenational Conference on Information Systems Analysis and Synthesis* (ISAS 98), vol 3

Parker E. Lynne, " Toward the Automated Synthesis of Cooperative Mobile Robot Teams", in *Proceedings of SPIE Mobile Robots XIII*, 1998, vol 3525

Parker E. Lynne, **1996**."On the Design of Behavior-Based Multi-robot Teams", in *Advantacd Robotics*, 10 (6)

Parker E. Lynne, **1994**." An Experiment in Mobile Robotic Cooperation", in *Proceedings of the ASCE Specialty Conference on Robotics for Challenging Environments*, febbraio

Phillips Peter, personal communication to Proudfoot, peter@phillips.net

Proudfoot K., 1998-1999. "RCX Internals. Technical Report", http://graphics.stanford.edu/kekoa/RCX

Sasuraibito, 1998 Internal pages Report to [Gasperi 98] http://www.geocities.co.jp/Technopolis/6264/mindstorms/internal.html, sasuraibito@geocities.co.jp

Scheier Christian e Lambrinos, **1995**."Adaptive Classification in Autonomous Agents", 9 ottobre

Smithers Tim, 1992."Personal comuniction to Stell Luc",Edinburg mobile robot group

Steels Luc, **1994a**."A Case Study in the Behavior-Oriented Design of Autonomous Agents", submitted to SAB-94

Steels Luc, **1994b**."Building Agents out of Autonomous Behavior Systems", in *The Artificial Life Route to Artificial Intelligence. Buidling Situated Embodied Agents*

Steels Luc, 1994c."When are Robots Intelligents autonomous agents?"

Steels Luc, "Mathematical Analysis of Behavior Systems"

Tsumura T., Fujiwara N., Shirakawa T. and Hashimoto M., 1981, "An Experimental System for Automatic Guidance of Roboted Vehicle Following the Route Stored in Memory", *Proc. Of the 11th Int. Symp. on Industrial Robots,* Tokyo, Japan

Villa Luis, 2000, "LegOS HOWTO 0.2.4", 22 Ottobre. luge@users.sourgeforge.net

Zuboff S., 1998."In the age of the Smart Machine", in *Basics books,* Inc., Publisher, New York

Altri Link

- CVS, http://sfdocs.sougeforge.net/sfdocs/
- Perl, http://www.activestate.com
- The official Lego Mindstorms Site. http://www.legomindstrorm.com
- The LegOS homepage. , http://noga.de/LegOS
- The *other*legOS homepage. This "official" page is used mainly for CVS and file download, since sourceforge provides 100M of free storage and unlimited downloads. It will (eventually) be the "official" site for all things legOS, including development and documentation., http://legos.sourceforge.net
- LUGNET- the Internet's home of all things Lego, http://www.lugnet.com
- the robotics mailing list , http://www.lugnet.com/news/display.cgi?/lugnet.robotics
- the LegOS mailing list, http://www.lugnet.com/news/display.cgi?/lugnet.robotics.rcx
- Lego Mindstorms Internals: This page has tons of links, focusing mainly on OS-type programs and communications protocols. http://www.crynwr.com/lego-robotics
- MIT Constructopedia: A work in progress at the MIT Media Lab. http://el.www.media.mit.edu/groups/el/projects/costructopedia
- O'Reilly Link Page: Jonathan Knudsen, writing for O'Reilly Publishing, has created a Mindstorms book. This page has all the links from the book. http://www.oreilly.com/catalog/Imstormresources/index.html
- Lego + Linux mini-HOWTO,http://linuxdoc.org/HOWTO/mini/Leg-HOWTO/
- NQC:http://www.enteract.com/~dbaum/cqc/index.tml
- RCX Arena Combat,http://www.azimuthmedia.com/RobotArena/mainframe.html

SIGLE

ADCSR	Status control register dell'A/D
ADI A/D	Converter interrupt
AFSM	FSM aumentata di pochi stati
AI	Artificial Intellignet
CIRA	Internaeational Symposium On Computational Intelligence In Robotics And Automation
CVS	Current Version System
DARS	Distribuite Autonomous Robotic-System
FSA	Finte State Automa/Machine/Acceptors
FSM	Macchina a stati finiti
IAS	Intellingent Autonomuos Systm
ICRA	International Conference Of Robotics And Automation
IR	Infra Red
IROS	International Conference of Intelligent Robot And System
ISAS	International Conference of Information System Analysis And Synthesis
LCD	Liquid cristal display
LegOS	lego O.S.
LNP	LegOS Network Protocol
NQC	Not Quite C
OS	Operative System
RAM	Random Access memory
RCX	Robot Command System
RCXCC	RCX Command Center
ROM	Read Only memory
RPM	Giri al minuto
SAB	International Conference on the Simulation Of Adaptative Robotics System
SKS	Scuola di Karatedo Shotokan