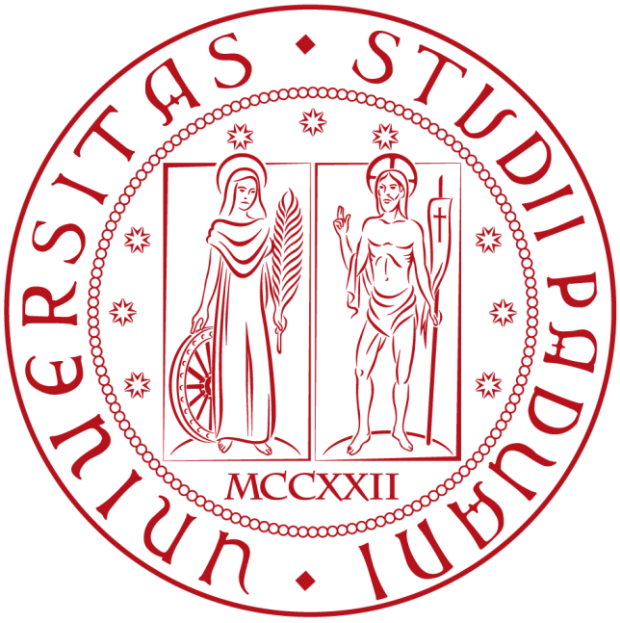
UNIVERSITA' DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
CORSO DI LAUREA IN INGEGNERIA INFORMATICA



SIMULAZIONE MEDIANTE ROBOT DIDATTICI MINDSTORMS : IL PUNTO DI VISTA DEL ROBOT GUIDA

Anno accademico
2009/2010

Relatore :
Prof. Michele Moro

Laureando :
Marco Guidi Colombi

Indice

1 Introduzione e Obiettivi della tesi

1.1 Obiettivi

L’obiettivo della tesi è stato quello di realizzare una simulazione della tipica manifestazione spagnola dell’Encierro, mediante l’utilizzo di robot LEGO Mindstorm NXT. In particolare l’analisi dei protagonisti di tale momento, e dei loro comportamenti ci ha permesso di strutturare opportunamente i robot, in modo da poter rendere la simulazione quanto più realistica possibile. Il lavoro svolto fa parte del progetto TERECOP attuato dalla Comunità Europea con lo scopo di introdurre l’utilizzo della robotica in ambiti didattici ed educativi. Si è cercato pertanto di sviluppare un sistema che prevede l’inseguimento autonomo di tre robot coscenti della realtà che li circonda, in grado di scambiarsi informazioni e pertanto di sincronizzarsi nei movimenti.

2 Encierro : partecipanti e svolgimento

2.1 La manifestazione

L'Encierro (letteralmente recinto), è una manifestazione tipica della Spagna, nata dalla necessità di trasferire i tori da fuori le mura all’arena, che consiste nel correre davanti ad un gruppo di tori, preceduti da buoi (cabestros) che conoscono il percorso e guidano la mandria. Il più famoso Encierro è senza dubbio alcuno quello che si svolge nella città di Pamplona, in Navarra, nei giorni tra il 7 ed il 14 luglio, in occasione delle feste di Sanfermines. Rappresenta l'evento centrale di tali feste che trasforma in uno spettacolo inimmaginabile in qualsiasi altro luogo del mondo, e proprio per questo è seguito da moltissimi turisti provenienti da diversi paesi al di fuori della Spagna.
L’Encierro si svolge solitamente nelle prime ore del mattino, in un percorso urbano che nei punti dove non delimitato da edifici viene per l'occasione recintato. Nelle manifestazioni maggiori si prevedono addirittura due recinti, di modo che gli spettatori si possano sedere sul recinto più esterno, mentre la zona compresa tra i due recinti viene usata come via di fuga dai corridori.
La possibilità di scendere in strada per partecipare all’evento è riservata ai maggiorenni, in alcuni casi si permette la partecipazione a chi ha compiuto i 16 anni.

I corridori che intendono prenderne parte si devono situare nel tratto compreso tra la piazza del Municipio e l'edificio detto de Educación, nella salita di Santo Domingo, prima delle 7,30 di mattina, dato che a quell'ora vengono chiuse le porte di accesso.

Nei minuti prima delle 8,00 i corridori sollevano il giornale che portano arrotolato per mantenere le distanze dal toro e cantano davanti ad una nicchia della salita di Santo Domingo contenente l'immagine di San Fermino, invocandone la benedizione.

La corsa comincia quando l'orologio della chiesa di San Cernin segna le otto in punto di mattino. Dopo il lancio di due razzi, i tori escono dal recinto per percorrere dietro i corridori gli 800 metri circa, che separano la porta dei recinti dall'arena di Plaza de Toros.

Mediamente impiegano circa tre o quattro minuti per effettuare l'intero percorso, anche se in occasioni eccezionali la corsa si è prolungata per oltre dieci minuti, a causa di alcuni tori rimasti "staccati" della mandria.

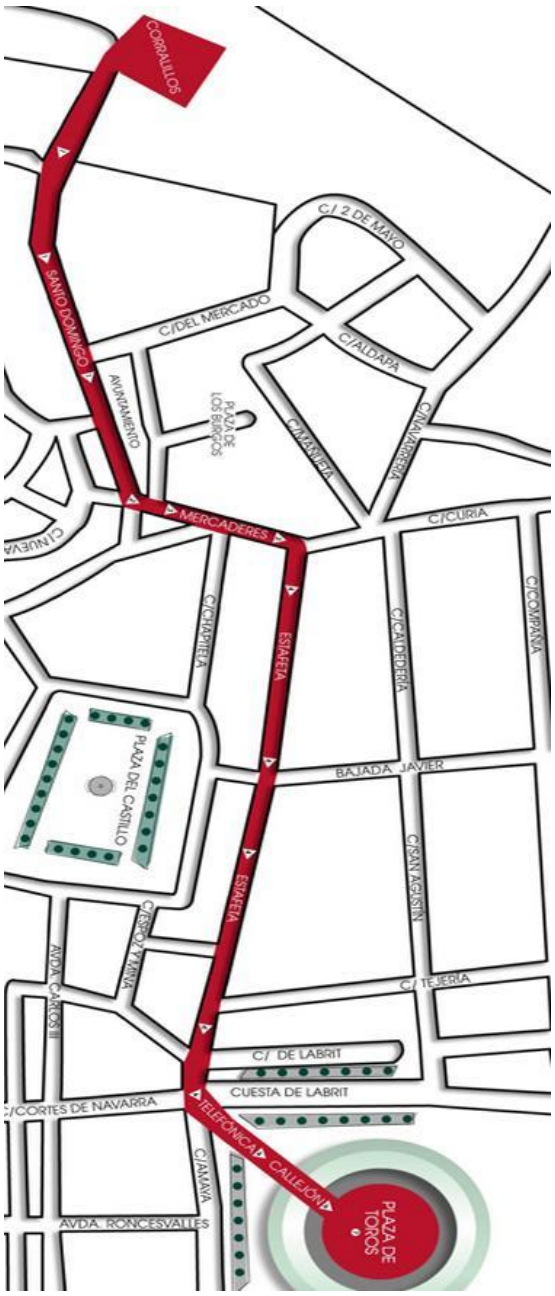


Figura 1 - Il percorso dell'Encierro di Pamplona

La tecnica per correre vicino ai tori per la maggior parte del tempo è partire piano e quando gli animali si avvicinano, cominciare a correre il più veloce possibile, quando il toro è troppo vicino si esce dal percorso; ogni corridore inoltre non deve incrociare la linea di corsa degli altri partecipanti. Per uscire dalla corsa è anche possibile arrampicarsi sul recinto piú interno o nascondersi in qualche pertugio tra le case.

2.2 Studio e analisi dei dati

Dalla descrizione riportata nel paragrafo precedente è facile estrapolare i dettagli fondamentali dell'Encierro, e poter cosi stabilire i ruoli interpretati dai robot utilizzati.
In particolare si distinguono tre personaggi che partecipano alla corsa :

- I *corridori* : coloro che scappano all'interno del percorso, e cercano di farsi inseguire dai tori, senza essere presi;

- I *cabestros* : ovvero i buoi che stanno sempre davanti alla mandria e fungono da guida per i tori in modo da non farli perdere;

- I *tori* : sono gli animali più giovani che, non conoscendo il percorso in questione, seguono i cabestros durante la corsa.

E' possibile inoltre suddividere l'intero percorso in tre zone principali :

- Il *recinto iniziale* : è il luogo dove viene posta la mandria prima dell'inizio della corsa, e da cui viene fatta partire;

- Il *percorso tra le vie cittadine* : è l'insieme delle vie che compongono il percorso, e nelle quali i corridori scappano inseguiti dalla mandria, percorrendo traiettorie casuali;

- L'*arena finale* : è il luogo dove al termine dell'Encierro gli animali vengono rinchiusi e lasciati vagare liberamente.

3 La Tecnologia NXT Mindstorms

3.1 Introduzione

Lego Mindstorms è una linea di prodotti LEGO che combinano mattoncini programmabili con motori elettrici, sensori, mattoncini LEGO, pezzi di LEGO Technic (come ingranaggi, assi e parti pneumatiche) per costruire robot e altri sistemi automatici e/o interattivi.
L'ultimo prodotto Mindstorms rilasciato è il Mindstorms NXT (agosto 2006).

Il kit è composto da :

- 519 pezzi LEGO Technic
- 3 servomotori, con sensore di rotazione integrato e feedback per il controllo di precisione
- 1 sensore tattile
- 1 sensore luminoso capace di rilevare intensità luminosa
- 1 sensore sonoro, con riconoscimento di tono e timbro
- 1 Sensore di movimento e distanza ad ultrasuoni
- 1 brick NXT.

Figura 2 - Lego Mindstorms NXT

3.2 Caratteristiche Tecniche

Il Mindstorms NXT possiede quattro porte di ingresso e tre di uscita, ma grazie all'implementazione dell'interfaccia digitale I2C, è possibile aumentarne il numero con dei moduli esterni. I connettori sono molto simili ai comuni RJ11. L'alimentazione è garantita da 6 batterie 1,5 Volt tipo AA oppure da una batteria ricaricabile (presente nel kit educational).

Le specifiche dell'NXT sono quindi le seguenti:

- Processore a 32 bit Atmel AT91SAM7S256 (classe ARM7) a 48 MHz
- Coprocessore 8 bit Atmel ATmega48 a 8 MHz, con 4k flash e 512 byte RAM
- 256 KB di memoria flash
- 64 KB di RAM
- Interfaccia bluetooth v2.0 (per trasferire il software o per controllare il robot da remoto)
- Display LCD bianco e nero da 100×64 pixel
- Può essere programmato su PC o Mac
- Speaker mono 8 bit fino a 16 KHz;
- Tastiera con quattro tasti in gomma.
- 4 porte di INPUT (ingressi 1 2 3 4) con connettore (RJ12 speciale)
- 3 porte di OUTPUT (uscite A B C) con connettore (RJ12 speciale)
- Interfaccia USB 2.0

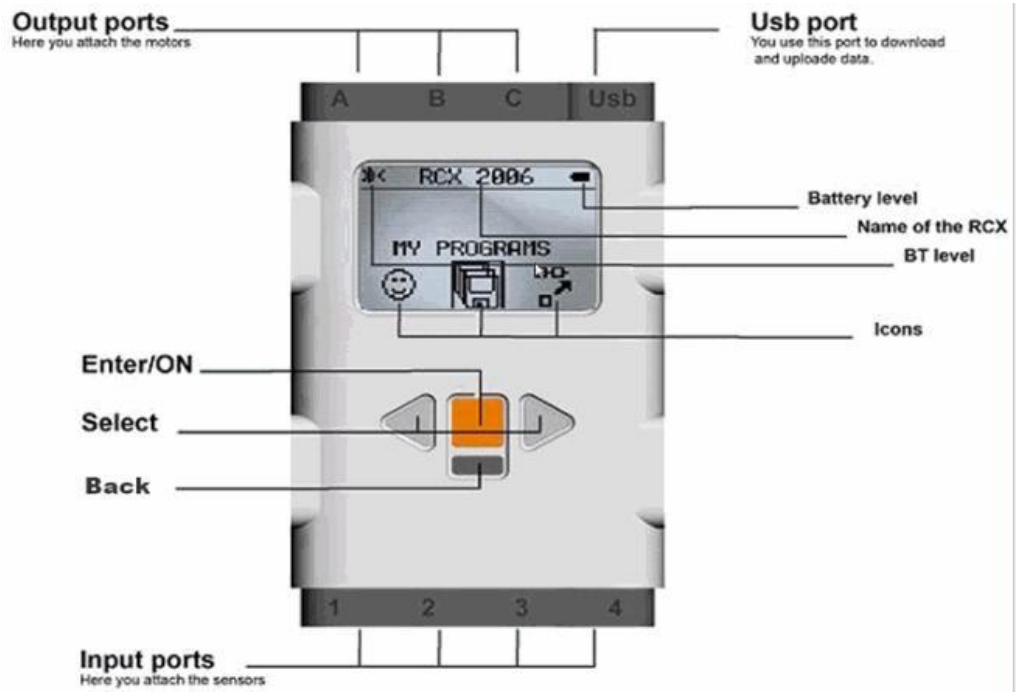


Figura 3 - Brick NXT

3.3 I Servomotori

Il kit fornito da LEGO comprende tre servomotori funzionanti in corrente continua (DC), ciascuno dei quali viene collegato a una delle 3 porte di OUTPUT (A, B o C).
Il servomotore ha al suo interno un sensore di rotazione per misurare sia la velocità che la distanza, che vengono trasmessi al processore del NXT. La possibilità di controllare accuratamente permette di sincronizzare più motori facendo in modo che si muovano alla stessa velocità. Inoltre con gli ingranaggi presenti nel Kit è possibile modificare ulteriormente il rapporto di trasmissione.

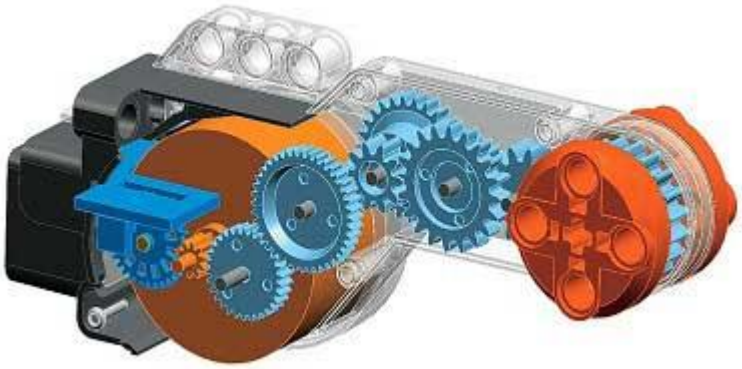


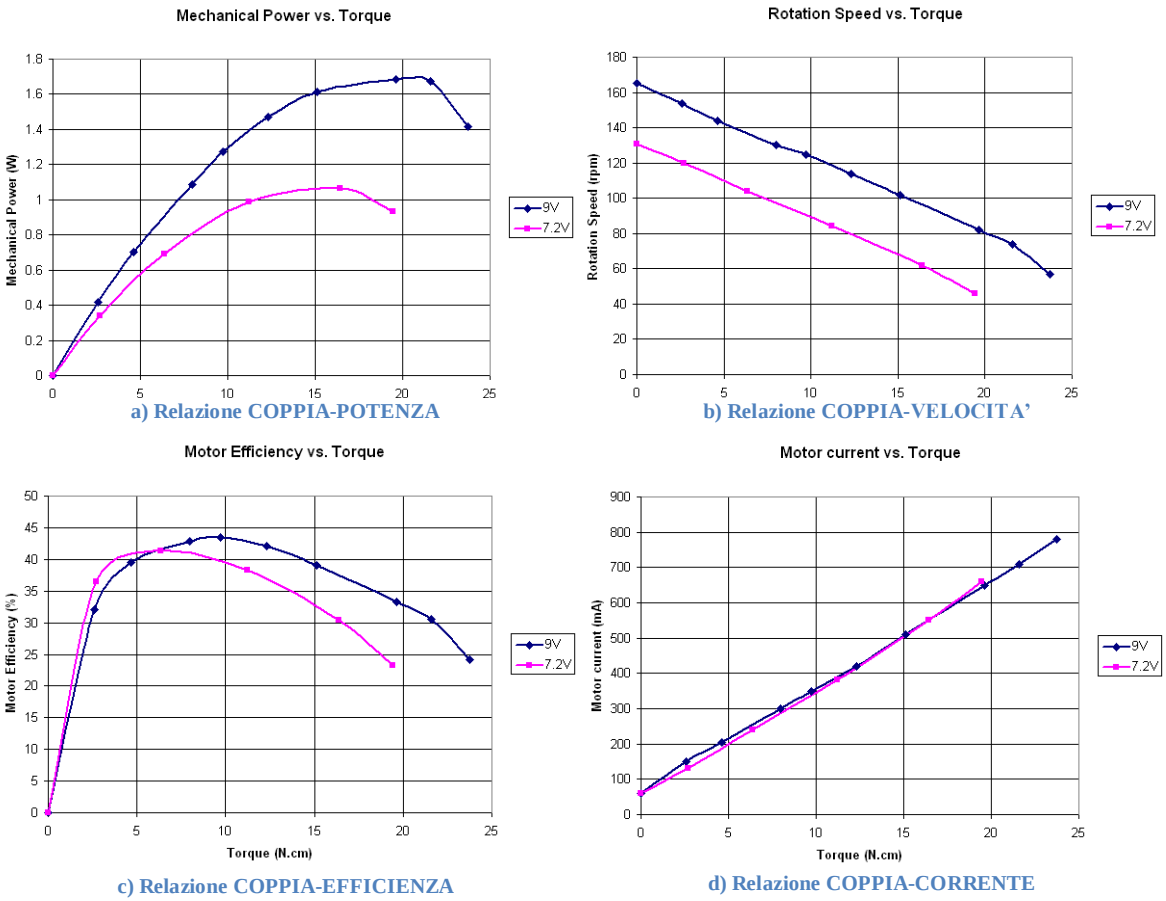
Figura 4 - Interno del servomotore

Figura 5 - Esterno del servomotore

Ogni servomotore presenta le seguenti caratteristiche:

- Tensione alimentazione 9V (DC)
- Velocità max 170 rpm (giri/minuto) (117 rpm a 9V)
- Potenza meccanica a 9V 2,03W
- Potenza elettrica a 9V 4,95W
- Efficienza a 9V 41%
- Assorbimento a 9V 0,55A
- No-Load current 60 mA
- Coppia a 9V 16,7 N*cm
- Coppia in stallo 50 N*cm
- Corrente di stallo 2 A
- Peso 80 gr.

Di seguito sono riportate le caratteristiche del motore NXT in funzione della coppia di carico, sia quando è alimentato con batterie alcaline (linea blu) sia quando è installata la batteria al litio presente nel kit (linea rosa).



a) Relazione COPPIA-POTENZA

b) Relazione COPPIA-VELOCITA'

c) Relazione COPPIA-EFFICIENZA

d) Relazione COPPIA-CORRENTE

Figura 6 - Caratteristiche Servomotore

3.4 I Sensori

Il kit LEGO Mindstorms NXT fornisce diversi sensori ciascuno dei quali è reso funzionante mediante il collegamento a una delle 4 porte di INPUT (1,2,3 o 4). In particolare sono presenti 4 tipi di sensori :

- Sensore di contatto (3.4.1)
- Sensore sonoro (3.4.2)
- Sensore ad ultrasuoni (3.4.3)
- Sensore di luce (3.4.4)

3.4.1 Sensore di contatto

E' un sensore molto semplice che permette al robot di avere il senso del tatto. Il segnale fornito è di tipo booleano ON-OFF (0 se rilasciato, 1 se premuto). E' tuttavia possibile settare il sensore per leggere il valore grezzo (RAW da 0 a 1023).

Figura 7 - Sensore di Contatto

3.4.2 Sensore sonoro

E' un sensore che conferisce al robot la capacità di sentire i suoni che lo circondano. Il segnale fornito è di tipo analogico ed è proporzionale al suono registrato.

Il sensore può misurare livelli di suono fino a circa 90 dB. Essendo molto complicato rilevare un valore assoluto, questo viene indicato in percentuale [%] (da 4-5% per rumore di fondo a 40-100% per musica ad alto volume). La lettura può essere in dB o dBA.

Figura 8 - Sensore Sonoro

3.4.3 Sensore ad ultrasuoni

Grazie a questo sensore (e al sensore di luce 3.4.4) il robot è in grado di avere una sorta di capacità visiva. Infatti esso è in grado di misurare la distanza da un oggetto posto nella sua direzione. Il segnale fornito è di tipo analogico ed è proporzionale alla distanza dell'oggetto rilevato. La misura effettuata viene espressa in centimetri (o pollici) da 0 a 255cm , con precisione di ± 3 cm.
Per effettuare tale misura, il sensore invia un onda sonora contro l'oggetto e ne calcola il tempo impiegato a tornare indietro.
Per tale ragione si può intuire come sia più sensibile a oggetti grandi con superfici dure, mentre la misura risulta meno accurata o addirittura inefficiente con con oggetti sottili, piccoli o con superficie non piana.

Figura 9 - Sensore ad Ultrasuoni

3.4.4 Sensore di luce

E' uno dei sensori più significativi per il robot (assieme al sensore ad ultrasuoni 3.4.3) se gli si vuole conferire una sorta di vista. Esso infatti permette di distinguere tra buio e luce, in quanto è in grado di misurare sia l'intensità della luce in una stanza che l'intensità della luce riflettente su una superficie. Il segnale fornito è di tipo analogico ed è proporzionale alla luminosità registrata. Mediante tale sensore è possibile riconoscere le diverse tonalità di grigio.
Una grossa problematica riscontrata nell'utilizzo di questo sensore è dovuto alla sua estrema sensibilità nella misura dell'intensità di luce. Infatti sono state riscontrate enormi differenze di lettura tra superfici lucide e opache. Inoltre l'utilizzo di tale sensore in ambienti fortemente illuminati risulta pressoché inutile in quanto non è in grado di percepire variazioni significative, fornendo costantemente il valore più alto di intensità luminosa.

Figura 10 - Sensore di luce

3.4.5 NXT Cam

A differenza dei sensori trattati precedentemente NXTCam è un sensore compatibile con i robot Lego Mindstorms NXT, non compreso nel kit e prodotto dalla *mindsensor.com* (acquistabile perciò dal sito internet dell'azienda). Si tratta di un motore di elaborazione realtime di immagine. Si può pensare ad esso come un sottosistema di visone con un processore on-board ed un protocollo di interfaccia che è accessibile attraverso le porte standard utilizzate per i normali sensori NXT. Quest' interfaccia offre informazioni ad alto livello dell'immagine che NXTCam vede.
Le informazioni processate contengono le coordinate della cornice rettangolare che racchiude gli oggetti di interesse nella visuale dell' NXTCam in modalità "Line Tracking".

Possono essere memorizzati fino ad un massimo di otto colori contemporaneamente.
Per le operazioni di runtime (dal robot, in modo autonomo), si connette la NXTCam al robot NXT attraverso una porta per sensori, utilizzando uno dei cavi standard presenti nel kit del Mindstorms.
Per quanto riguarda invece le operazioni offline, per programmarla e configurarla, si connette la NXTCam al PC utilizzando un cavo mini-USB.

Figura 11 - NXT Cam

La NXTCam fornisce le seguenti possibilità:

- Tracciamento di al più 8 oggetti differentemente colorati a 30 frames/sec.
- Configurazione della NXTCam usando l'interfaccia USB su Windows XP, Windows Vista.
- Supporto di due modalità di tracciamento: Object tracking and Line tracking.
- Supporto di statistiche in tempo reale sugli oggetti tracciati (numero di oggetti, colore degli oggetti, coordinate della cornice o della linea) attraverso la porta NXT standard del sensore.
- Risoluzione dell'immagine tracciata di 88 x 144 pixels at 30 frames/second.
- Massima risoluzione di 176 x 144 pixels dell immagine colorata sul PC via porta USB.
- Massimo consumo di potenza (42 mA at 4.7 V).
- Utilizza il protocollo di comunicazione I2C compatibile l'NXT.
- Supporta la Auto Detecting Parallel Architecture (ADPA) per l'NXT sensor bus. Questo significa che la NXTCam può coesistere con sensori digitali di LEGO o di terzi sulla stessa porta NXT.
  Il supporto di ADPA permette all'utente di impiegare più sensori nella stessa porta senza la necessità di un multiplexer esterno, riducendo le dimensioni totali senza compromettere le funzionalità.

Per configurare la telecamera viene messo a disposizione un software chiamato NXTCamView che facilita le impostazioni e la memorizzazione dei colori prescelti nella memoria del sensore.

Di seguito è riportata una videata del software NXTCamView, nella quale si possono osservare gli oggetti di interesse e le loro informazioni di monitoraggio. Più precisamente nella finestra in alto a sinistra si nota il campo di visione di NXTCam. Gli oggetti di interesse da questa angolazione sono le penne rosse e blu.
Si può inoltre notare quanto detto in precedenza, infatti nella parte destra della schermata sono presenti le coordinate dei contorni dei rettangoli rilevati.

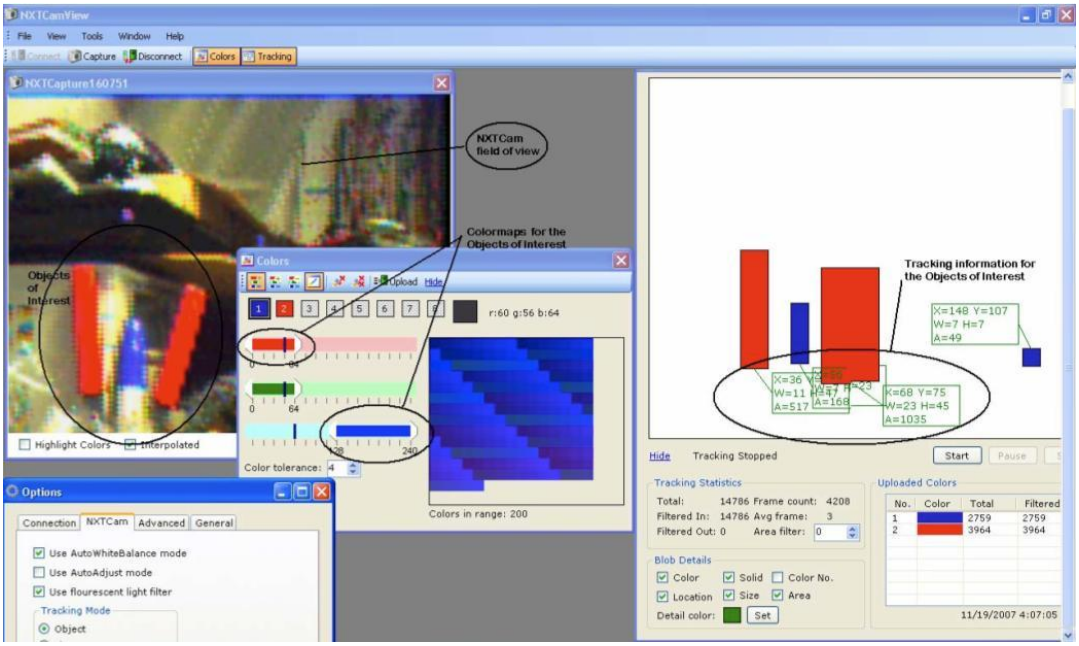


Figura 12 - NXTCam in modalità ObjectTracking

Nella modalità in Line tracking, vengono restituite le coordinate del punto di partenza e del punto d'arrivo della linea che rappresenta l'oggetto analizzato. Nella figura seguente, i riquadri sono disegnati tramite la linea definita dalle coordinate ricevute dal NXTCam.

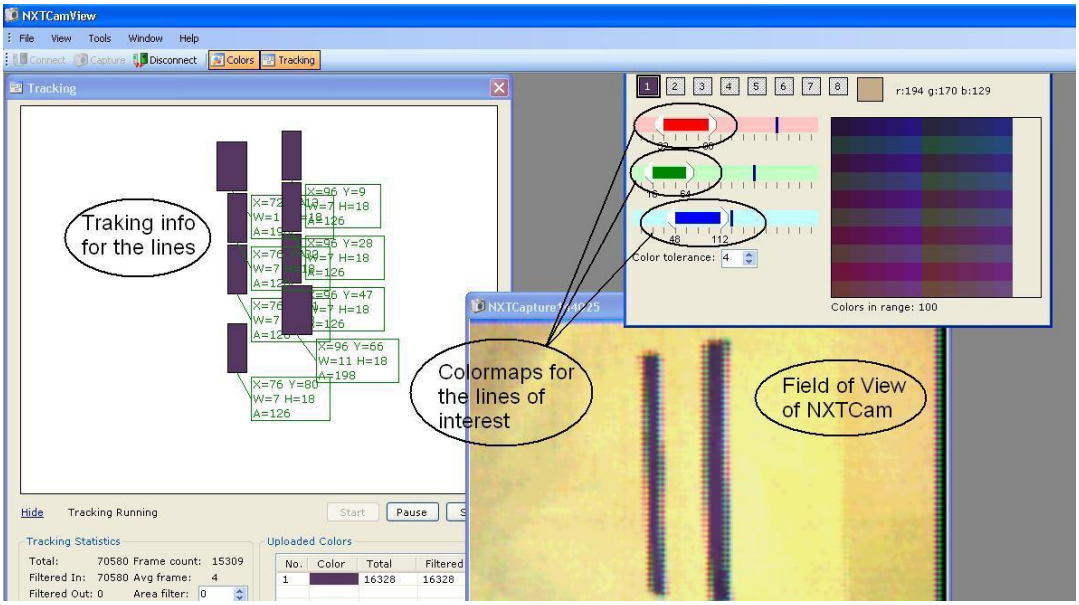


Figura 13 - NXTCam in modalità LineTracking

4 Descrizione della simulazione

4.1 Robot coinvolti

In seguito all'analisi effettuata sulla manifestazione dell'Encierro (riportata nel paragrafo 2.2) abbiamo scelto di strutturare la simulazione prevedendo tre robot svolgenti i seguenti compiti :

- Un *robot corridore* (4.1.1) : rappresenta uno qualunque dei corridori partecipanti alla corsa, perciò dovrà essere in grado di scappare dagli inseguitori, variando la propria velocità a seconda della distanza da questi.

- Un *robot master* (4.1.2) : rappresenta uno dei cabestros situati davanti alla mandria. Il suo compito è quello di inseguire il corridore cercando di raggiungerlo.

- Un *robot slave* (4.1.3) : rappresenta uno dei tori più giovani, pertanto dovrà seguire fedelmente il robot master replicandone i movimenti.

Analizziamo ora nel dettaglio le caratteristiche dei robot coinvolti.

4.1.1 Robot Corridore

Per svolgere correttamente il suo compito, il robot corridore dovrà seguire una linea di nastro isolante nero, tracciata sul piano di appoggio a formare un percorso, e ben distinguibile dal colore del piano stesso.
Inoltre sarà in grado di rilevare eventuali avvicinamenti da parte degli inseguitori, e di conseguenza varierà la propria velocità scappando da questi. Infine, proprio come i toreri, esso sarà ricoperto di un guscio di colore rosso per esser cosi facilmente individuato dagli inseguitori.

Per quanto riguarda le caratteristiche dettagliate di questo robot, verranno trattate più avanti nel capitolo 5.

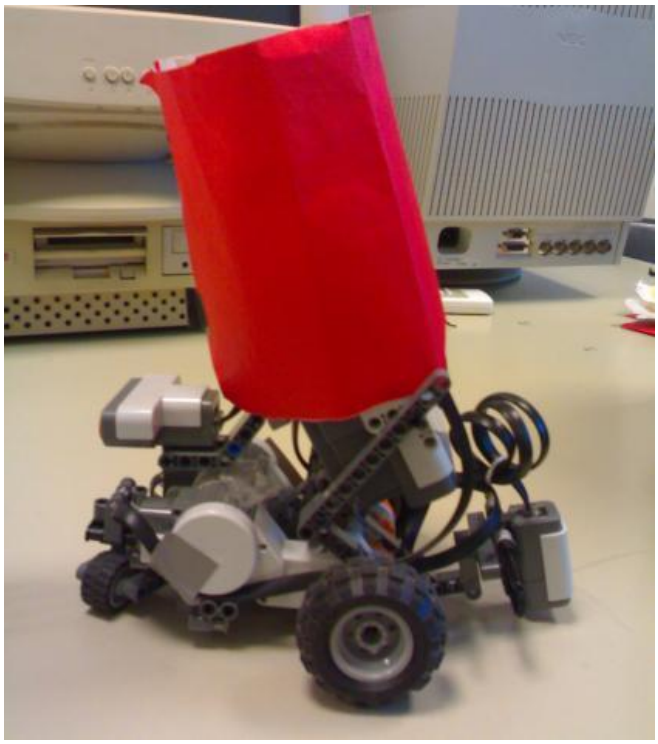
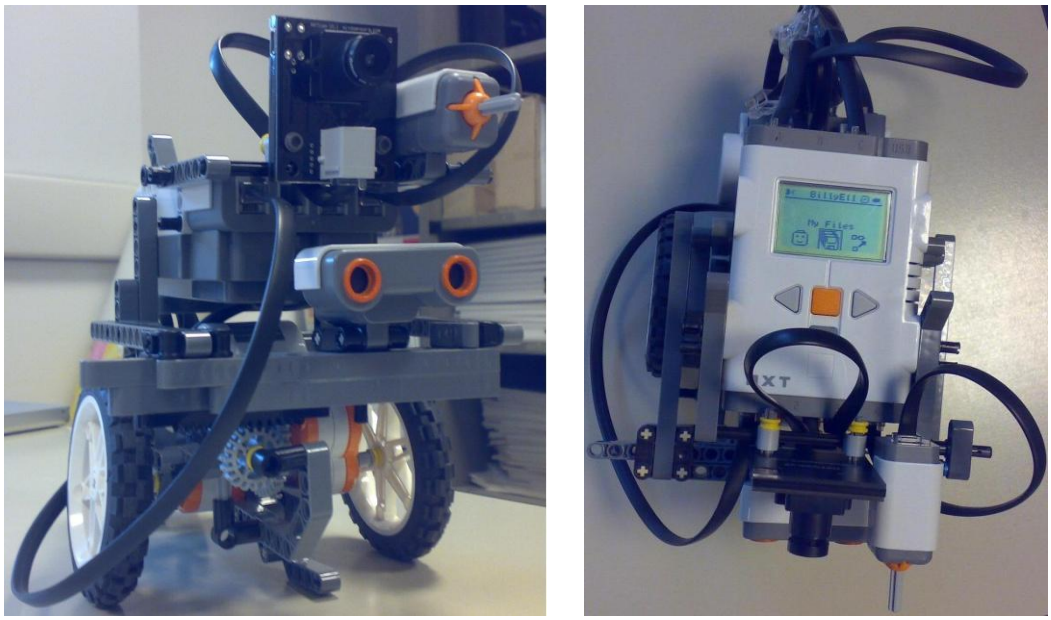


Figura 14 - Robot corridore

4.1.2 Robot Master

Tra i tre, questo robot è il più articolato strutturalmente in quanto presenta per prima cosa una sorta di testa mobile, costituita da un sensore ad ultrasuoni in grado di scorrere orizzontalmente su di una corsia (Figura 15a).



a) b)

Figura 15 - Robot master

L’utilizzo di tale sensore unito ad un meccanismo di spostamento è stato dettato dalla volontà di voler rendere il robot capace di effettuare le curve, servendosi delle leggi trigonometriche note.
Questo robot è inoltre dotato della sopracitata NXTCam (3.4.5), grazie alla quale è in grado di localizzare il robot corridore e variare i propri movimenti a seconda degli spostamenti di quest’ultimo.
Una volta effettuati, tali movimenti vengono trasmessi al robot slave, grazie ad una connessione bluetooth stabilita tra i due robot.
Infine, per rendere più reale la simulazione, abbiamo deciso di installare un sensore di tatto nella parte frontale del robot, in modo tale da avere un riscontro nell’istante in cui riesce a “incornare” il corridore (Figura 15).

4.1.3 Robot Slave

Il robot slave ha una struttura standard in quanto prevede esclusivamente l’utilizzo di un sensore ad ultrasuoni per effettuare il posizionamento iniziale (Capitolo 4.3). L’idea di incorporare un tale sensore può permettere eventualmente degli sviluppi futuri, prevedendo ad esempio un percorso con presenza di ostacoli che il robot dovrà evitare.
Il resto dei movimenti sono effettuati replicando perfettamente il percorso compiuto dal robot master. Difatti grazie all’interfaccia bluetooth questi due robot si scambiano informazioni relative ai movimenti effettuati (espressi in termini di rotazione e velocità di rotazione dei motori).

Figura 16 - Robot slave

4.2 Ambiente di svolgimento dell'esperienza

Per lo svolgimento della simulazione abbiamo avuto a disposizione un laboratorio dotato di illuminazione (a neon) e di due tavoli.
Abbiamo pertanto deciso di disporre i due tavoli a formare una "Elle" (Figura 17), in modo tale da far assomigliare il percorso quanto più possibile all'originale di Pamplona (2.1).
Su una porzione di uno dei due tavoli abbiamo costruito una sorta di recinto iniziale di cartone, dove i due inseguitori vengono posti per la partenza della simulazione (Figura 18). Inoltre la volontà di rendere il percorso più simile possibile alle strade cittadine, ci ha spinto alla costruzione di "mura" di cartoncino, delimitanti il percorso stesso (Figura 20).

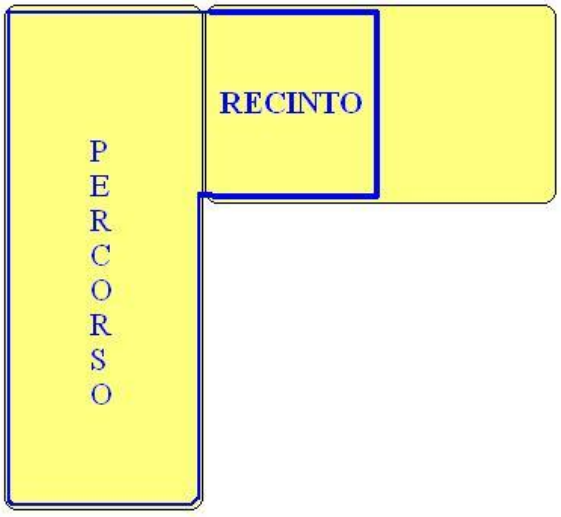


Figura 17 - Disposizione tavoli

Il percorso che il "corridore" potrà seguire ,è formato da linee di nastro isolante nero, stese sul tavolo ad indicare le varie direzioni possibili.
In aggiunta sono stati previsti all'interno del percorso dei "bivi" che consentono al corridore di scegliere casualmente la direzione da prendere.
Per l'identificazione del bivio sono stati disposti, prima della diramazione stessa (sopra il nastro), dei "marker" di colore giallo (Figura19).

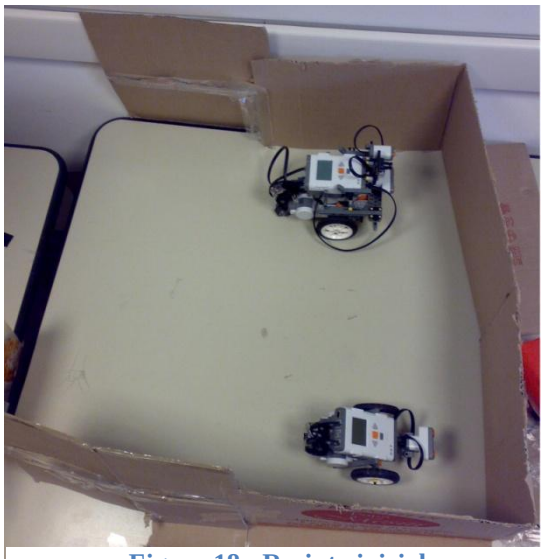


Figura 18 - Recinto iniziale

La scelta di questo colore è stata dettata da numerose prove effettuate, dalle quali è risultato essere il colore meglio distinguibile dal sensore di luce presente sul robot corridore (maggiori dettagli sono presenti nelle considerazioni del paragrafo 5.2.3).

Figura 19 - Marker giallo che identifica il bivio

Figura 20 - Mura di cartoncino che delimitano il percorso

Come si può osservare dalla Figura 21, abbiamo anche deciso di introdurre una variante rispetto al percorso pamplonico, in modo tale da rendere la simulazione di durata maggiore e permettere di osservare nel dettaglio il comportamento dei tre robot in situazioni diverse.

In particolare il poco spazio a disposizione ha portato a non prevedere un'arena finale, bensì a strutturare un percorso circolare che riconduce al recinto iniziale.

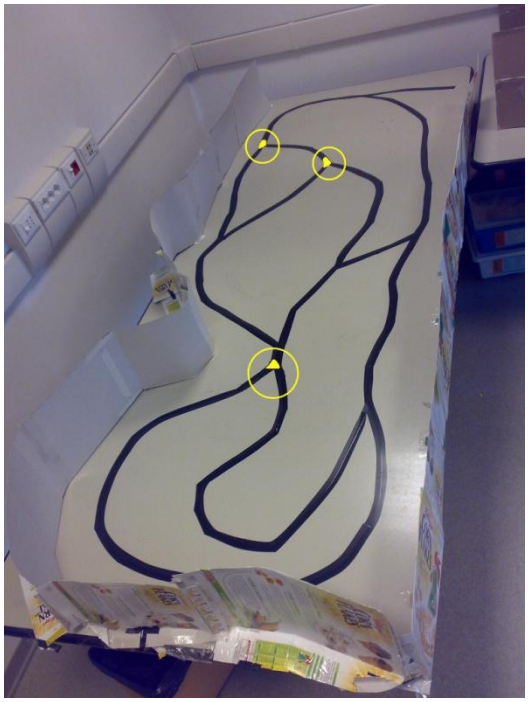


Figura 21 – Percorso di nastro nero (in giallo si osservano i marker sopracitati)

4.3 Svolgimento della simulazione

La simulazione, cosi come l'Encierro, prevede inizialmente che gli inseguitori (robot master e slave) vengano posti dentro al recinto, mentre il corridore si posizioni appena fuori e sia pronto a scappare non appena la corsa avrà inizio. Per tale ragione è stata prevista una prima fase, che possiamo appunto definire "iniziale", durante la quale i tre robot si allineano sincronizzandosi per la partenza della corsa.

Di seguito analizziamo in dettaglio in che modo si svolge tale fase :

a. Per prima cosa è necessario posizionare correttamente i due robot inseguitori all'interno del recinto. Un posizionamento corretto prevede prima di tutto che vengano posti con il sensore a ultrasuoni in direzione della parete B del recinto (Figura 22). Inoltre è fondamentale che essi non siano perfettamente allineati, bensì che uno dei due si trovi più avanti rispetto all'altro (almeno di 20cm), in modo tale da non essere d'intralcio al sensore di ultrasuoni durante la misurazione. Questa condizione deriva dalla funzione di autodeterminazione della propria posizione all'interno del recinto, implementata nell'algoritmo per la fase iniziale.

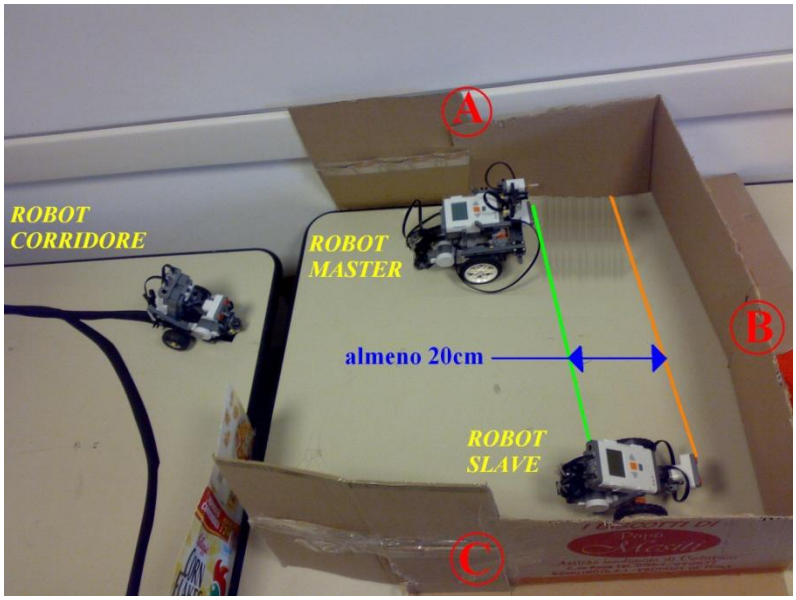


Figura 22 – Posizionamento iniziale CORRETTO

b. E' necessario a questo punto controllare che sia attiva la connessione bluetooth tra i tre robot in modo tale da permettere loro di scambiarsi i dati necessari.

c. Viene quindi eseguito il programma del robot master, il quale prevede due rotazioni di 90 gradi al termine di ciascuna delle quali vengono eseguite le misurazioni di distanza dalle pareti del recinto (B e C). Si esegue dunque il programma del robot slave.

d. Determinata la propria posizione, il robot master ha il compito di inviare le proprie coordinate al robot slave, il quale, a questo punto è in grado di calcolare le componenti del movimento, necessario per giungere alla posizione occupata dal robot master. Una volta giunto in tali coordinate, esso può replicare pari passo gli spostamenti effettuati dal robot che lo precede. Infine viene segnalato al corridore, tramite un messaggio bluetooth, che può iniziare liberamente la sua corsa sul percorso segnato.

Portata a termine questa prima fase ha inizio la vera e propria corsa. Il corridore comincia a percorrere il percorso seguendo il tratto nero grazie al sensore di luce. Essendo ricoperto da un cilindro rosso è facilmente individuabile dall'inseguitore dotato di "vista" (robot master).
L'algoritmo di inseguimento, presente in quest'ultimo, prevede l'attivazione di due task eseguiti per tutta la durata della simulazione :

- *Camera* : si occupa dell'inseguimento del corridore, quando il cilindro rosso di cui è ricoperto, è tracciabile dalla NXTCam;

```
task camera(){
      while (true){
            Acquire (semo)
            if (nblobs >0){ // Corridore "agganciato"
                  // CODICE DI GESTIONE DELLA TELECAMERA
            }else { // Corridore non individuato
                  nofound++;
                  if (nofound>3)
                        start_testa=true;
            }
            Release(semo);
      }
}
```

Codice del task camera

- *Sonar* : gestisce il movimento della "testa mobile" quando non è possibile "agganciare" il corridore , in modo tale da permettere al robot di proseguire all'interno del percorso;

```
task sonar (){
      while (true){
            Acquire(semo);
            if(start_testa){
                  // CODICE DI GESTIONE DEL SONAR
            start_testa=false;
      }
      Release (semo);
}
```

Codice del task sonar

Dai segmenti di codice allegati è possibile osservare che il task *camera* ha una priorità maggiore rispetto a *sonar* in quanto si presume che la telecamera riesca ad "agganciare" il corridore per la maggior parte della simulazione. Pertanto il controllo non viene ceduto direttamente al task secondario, bensì sono necessari tre cicli in cui la NXTCam non riesca ad individuare il cilindro rosso per cedere il controllo al task *sonar*.
Per quanto riguarda il robot slave, una volta raggiunta la posizione inizialmente occupata dal master, replica esattamente i movimenti effettuati da quest'ultimo e trasmetti tramite la connessione stabilita.

4.4 Problemi riscontrati

In questa parte della simulazione abbiamo riscontrato principalmente due problemi che condizionano la buona riuscita dell'esperienza.
Il primo problema è rappresentato dalla scarsa precisione del sensore ad ultrasuoni in dotazione, che alcune volte porta il robot a non riconoscere le pareti attorno al percorso. Per risolvere tale problema basterebbe effettuare un numero sufficiente di misurazioni con il sonar, tuttavia i tempi di reazione del robot diverrebbero notevoli, e ai fini della simulazione compromettenti. Avendo a disposizione spazi più ampi e un'ambientazione diversa, si potrebbe considerare questa miglioria.
Si è osservato inoltre un secondo problema, ovvero un'anomala convergenza a sinistra del robot slave. L'errore è minimo se considerato singolarmente, tuttavia a lungo andare si propaga incidendo sulla traiettoria che il robot slave dovrebbe compiere.

Dopo numerose prove che comprendono addirittura la sostituzione del servomotore (oltre a ruote e cablaggi), siamo giunti alla conclusione che il problema potesse derivare da una non equivalente alimentazione su tutte le porte del brick, influenzando in tal modo la velocità di rotazione dei motori stessi.

4.5 Accorgimenti utili per una corretta simulazione

Considerati i problemi descritti nel paragrafo precedente (4.4), è importante tenere presente dei piccoli accorgimenti in maniera tale da ridurne l'influenza sulla simulazione.
Per prima cosa è consigliabile che le batterie dei tre brick siano perfettamente cariche prima di avviare la corsa. In alcuni casi infatti ,una carica non completa, ha contribuito a far ruotare i due motori con velocità differenti e a percorrere pertanto traiettorie curvilinee non previste.
Un'importante accorgimento riguarda la costruzione dell'ambiente di svolgimento della simulazione. Più precisamente è necessario che le "mura" di cartoncino poste attorno al percorso siano di altezza almeno di 20cm in modo tale da poter essere individuate dai sonar presenti sui tre robot. Si osserva inoltre che pareti quanto più regolari possibili, contribuiscono a rendere efficienti le rilevazioni effettuate dai sonar, evitando riflessioni anomale.
E' utile infine tenere presente un'accortezza riguardo la NXTCam.
In particolare è consigliabile controllare il tracking del cilindro rosso prima di iniziare una nuova simulazione (ed eventualmente ritracciare il colore), in quanto è fortemente influenzato dalla condizioni di luce presenti nella stanza.

5 Robot Corridore

Il robot corridore ha il compito principale di seguire il percorso stabilito, formato da una linea di nastro isolante nero tracciata sulla superficie del tavolo. Dev'essere inoltre identificabile dagli inseguitori, per questo è completamente rivestito da un cilindro "rosso".
Una volta posizionato sul punto di inizio della traccia (appena fuori il recinto), resta in attesa di una segnalazione da parte del robot master. Quest'ultimo, grazie ad una connessione bluetooth stabilita tra i robot, appena effettuata la fase iniziale (descritta in 4.3) provvederà a segnalare che è pronto all'inseguimento, permettendo cosi al corridore di iniziare la propria fuga.
Ai fini della realisticità della simulazione, sono state inoltre previste delle biforcazioni, a indicare le diverse traiettorie che il corridore in fuga può prendere. Esso è infine in grado di monitorare la situazione alle sue spalle e agire di conseguenza : dovrà accelerare la corsa in caso gli inseguitori siano troppo vicini , rallentarla quando si allontanano e fermarsi se non rileva nulla attorno.

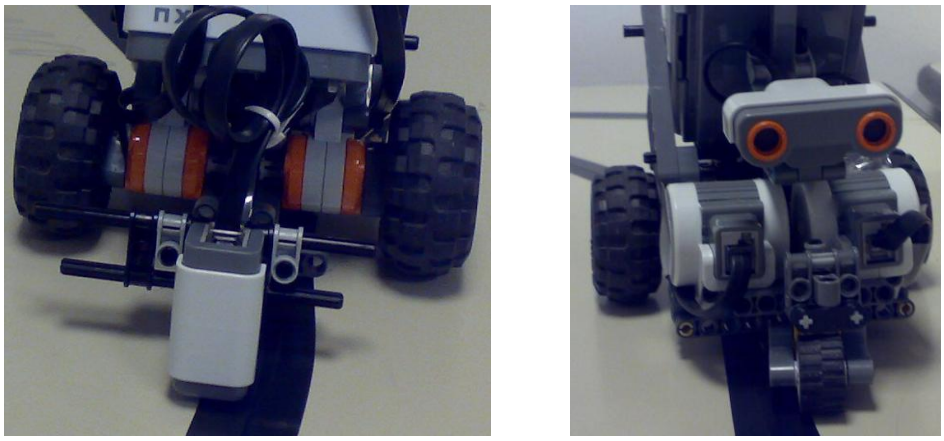
5.1 Scelta della struttura

Le premesse descritte permettono di comprendere le scelte effettuate in merito alla struttura del robot corridore.
In particolare si è scelto di dotare quest'ultimo con i seguenti sensori :

- Un *sensore di luce* : grazie al quale è in grado di distinguere la linea nera rispetto alla superficie del tavolo. Mediante tale sensore avviene anche l'identificazione dei bivi presenti;

- Un *sensore ad ultrasuoni (sonar)* : attraverso il quale è cosciente della distanza dalla mandria.

I due sensori sono posti specularmente infatti il primo è posto nella parte frontale del robot, mentre il secondo posteriormente.
Dopo numerose prove effettuate, si è deciso di posizionare il sensore di luce ad una altezza minima rispetto al piano del tavolo (circa 3mm), in modo tale da ridurre le riflessioni di luce provenienti dall'esterno.
Si è visto infatti che il sensore è estremamente sensibile all'illuminazione dell'ambiente in cui viene utilizzato, compiendo in alcuni casi misurazioni scorrette che lo portano ad allontanarsi dalla linea nera.



a) b)

Figura 23 – Posizionamento dei sensori nel robot corridore

Per avere la certezza di rilevare gli eventuali inseguitori prossimi all' "incornata" è stato installato il sensore ad ultrasuoni nell'estremità opposta, ad un altezza media rispetto a quella del robot e parallelamente al piano.

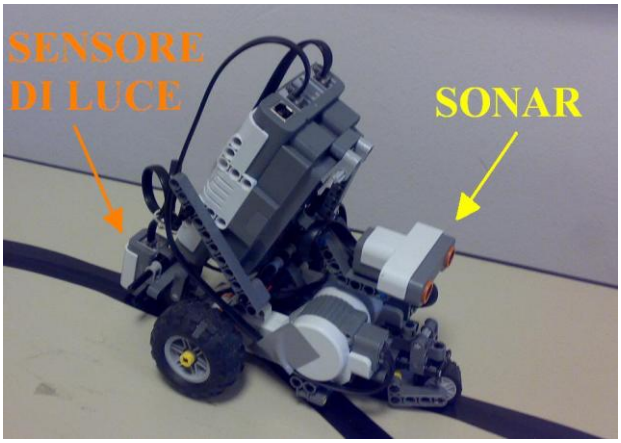


Figura 24 – Robot corridore

Nella Figure 23 e 24 si possono osservare le posizioni dei due sensori.

Il brick del corridore è stato ricoperto da un cilindro rosso di carta che permette alla NXTCam di identificarlo univocamente e inseguirlo.
Anche in questo caso la scelta del colore è stata determinata da varie prove effettuate che hanno delineato il rosso come colore che non si confonde con lo sfondo del percorso.

5.2 Analisi del codice NXC

Il codice completo del programma è allegato alle ultime pagine della tesi. Di seguito verranno trattate nel dettaglio le porzioni di codice più significative, utili per comprendere a pieno il funzionamento della simulazione.
Per la programmazione dei robot si è scelto NXC, un linguaggio basato su C/C++ e identico a quest'ultimo per la maggior parte dei costrutti e strutture. Pertanto per una corretta comprensione è necessario godere di una conoscenza base del linguaggio C/C++, e conoscere i metodi per il controllo dei robot, messi a disposizione da NXC (controllo rotazione motori, acquisizione input dei sensori, ecc.).

Per una più chiara comprensione del codice implementato nel corridore, possiamo pensarlo suddiviso in 4 parti principali :

- *Avvio connessione bluetooth e attesa segnale di partenza* (5.2.1)
- *Algoritmo FollowLine* (5.2.2)
- *Riconoscimento del bivio e scelta della direzione* (5.2.3)
- *Rilevamento distanza dagli inseguitori* (5.2.4)

5.2.1 Avvio connessione bluetooth e attesa segnale di partenza

Avendo a disposizione una connessione bluetooth risulta utile ed efficace prevedere che i tre robot si autoregolino per la partenza della corsa.
In questo modo si evita di dover attivare manualmente ciascun robot dal brick, oltretutto nel giusto istante di tempo. Per stabilire la suddetta connessone è sufficiente includere l'istruzione `BTCheck(BT_CONN)`, dove " `BT_CONN` " rappresenta il canale, e dovrà essere lo stesso per tutti i robot coinvolti.
Le seguenti righe di codice consentono di realizzare un ciclo di attesa nel quale si aspetta un messaggio di conferma (" a ") prima della partenza.

```
while(conferma!="a"){
     ReceiveRemoteString(INBOX,true,conferma);
}
```

Codice di attesa per la partenza

Il metodo `ReceiveRemoteString` consente di ricevere il dato inviato dal master e verificarne la correttezza. I parametri di questo metodo rappresentano in ordine la mailbox d'ingresso (in cui inserire i dati), l'abilitazione alla sovrascrittura e la variabile dove memorizzare il dato. E' necessario che i tre robot abbiamo valori di mailbox siano univoci, inoltre la mailbox d'uscita di chi invia il dato deve corrispondere a quella in ingresso di chi lo riceve.
La ricezione da parte del corridore di una stringa equivalente a quella predefinita (" a ") gli permette di terminare il ciclo e proseguire con il codice riguardante il movimento lungo la linea nera.

5.2.2 Algoritmo FollowLine

L'algoritmo implementato consente al robot di percorrere il percorso lungo la linea nera tracciata sulla superficie del tavolo.
Più precisamente il programma è costruito in modo tale da seguire il bordo destro del nastro. Sono definiti come prima cosa i due estremi dell'intervallo dei valori considerati "validi", cioè corrispondenti ad una posizione corretta del robot. La lettura di un valore compreso in tale intervallo permette di affermare che ci si trova sopra il bordo della linea e non è necessaria una variazione di direzione.
Nel nostro caso l'intervallo è formato dai seguenti estremi :

```
int Threshold1=570;
int Threshold2=590;
```

Naturalmente minore è il ΔT (ΔT=Threshold2-Threshold1), ovvero quanto più stretto è l'intervallo stabilito, tanto più alta è la precisione dell'algoritmo. Tuttavia questa considerazione teorica si è rivelata non soddisfacente a causa della poca precisione del sensore di luce, ma soprattutto all'altissima influenza che l'illuminazione ambientale ha sul sensore stesso. Per questo motivo l'intervallo scelto è risultato il miglior compromesso per una buona realizzazione della simulazione.
E' inoltre giusto precisare le seguenti informazioni :

Descrizione	**Porta del brick utilizzata**
Motore Destro	OUT_A
Motore Sinistro	OUT_C
Sensore di luce	IN_3
Sensore ad ultrasuoni	IN_4

La prima delle seguenti istruzioni inizializza la porta IN_3 del brick, specificando che vi è collegato un sensore di luce con luce attiva (IN_TYPE_LIGHT_ACTIVE). La seconda istruzione specifica la modalità di utilizzo "IN_MODE_RAW" del sensore, in base alla quale il valore restituito è un intero compreso tra 0 e 1023.

```
SetSensorType(IN_3, IN_TYPE_LIGHT_ACTIVE);
SetSensorMode(IN_3, IN_MODE_RAW);
```

Definito inoltre `SV = SensorRaw(IN_3)` il valore in ingresso letto dal sensore di luce, va specificato che *maggiore è tale valore, minore è la quantità di luce riflessa sulla superficie*. Di conseguenza una gradazione più scura corrisponde al un valore più elevato di SV rispetto ad una gradazione più chiara.

L'algoritmo "FollowLine" è il seguente :

```
// imposta la velocità del motore A (Destro)
if (SV < Threshold2){
      OnFwd(OUT_A, SpeedFast);
}
else {
      OnFwd(OUT_A, SpeedSlow);
}

// imposta la velocità del motore C (Sinistro)
if (SV > Threshold1){
      OnFwd(OUT_C, SpeedFast);
}
else {
      OnFwd(OUT_C, SpeedSlow);
}
```

Si può facilmente osservare che possono verificarsi tre diversi casi :

I. *Threshold1 < SV < Threshold2*

   Questa condizione rappresenta la corretta posizione del corridore rispetto alla linea. Significa in pratica che il robot sta seguendo perfettamente il margine destro della traccia di nastro nero.
   In questo caso le velocità di entrambi i motori non subiscono variazioni e restano impostate sulla massima velocità (*SpeedFast*), in modo da far percorrere una traiettoria rettilinea al corridore (Figura 25a).

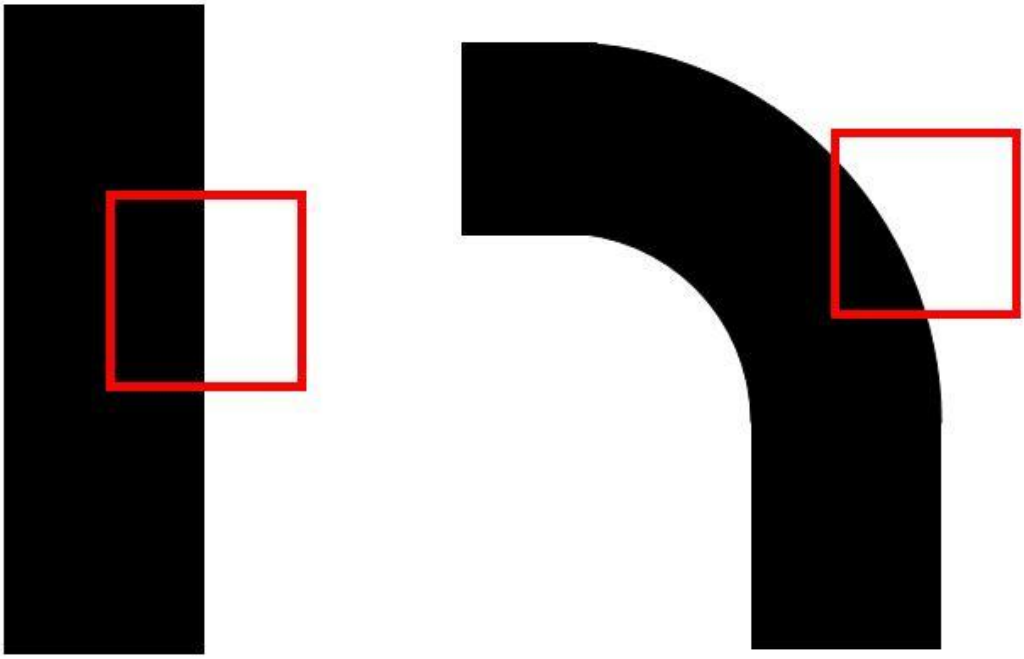
II. *SV* < *Threshold1*

Tale condizione si verifica in presenza di una curva verso sinistra. La lettura effettuata dal sensore indica un aumento dell'intensità luminosa dovuta all'uscita di curva da parte del robot(Figura 25b). Dovrà perciò essere apportata una correzione della direzione, in questo caso diminuendo la velocità del motore sinistro (*SpeedSlow*) rispetto al destro (*SpeedFast*).

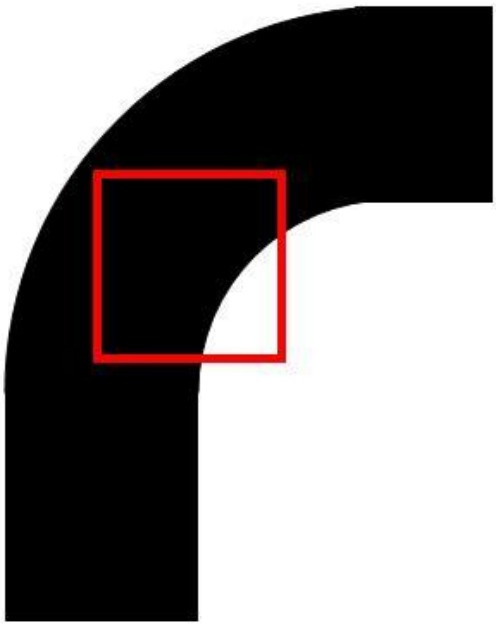
III. *SV* > *Threshold2*

Quest'ultima condizione, analogamente al caso precedente si manifesta in presenza di una curva verso destra. Una chiara dimostrazione è la diminuzione di intensità luminosa rilevata dal sensore dovuta all'attraversamento del nastro nero (Figura 25c).
In questo caso è necessario correggere la rotta del corridore diminuendo la velocità del motore destro (*SpeedSlow*) rispetto al sinistro (*SpeedFast*).

Seguono le figure rappresentanti i tre casi appena trattati.



a) **b)**



c)

Figura 25 - FollowLine : tre casi possibili

L'algoritmo descritto è posto all'interno di un ciclo *while(true)* in modo da ripetere le stesse operazioni all'infinito a meno di una terminazione forzata generata dal rimanente codice presente. In questo modo permette al corridore di percorrere tranquillamente l'intero percorso proposto.
E' necessario tuttavia implementare un'ulteriore algoritmo per la gestione dei bivi presenti lungo la strada e non considerati finora.

5.2.3 Riconoscimento del bivio e scelta della direzione

Il codice proposto di seguito permette al robot di riconoscere i bivi presenti lungo il suo percorso e decidere di conseguenza quale strada prendere. Prima di procedere all'analisi dell'algoritmo è doveroso fare alcune precisazioni in merito al marcatore utilizzato.
Sono state necessarie diverse prove per determinare il marker più adatto per questa simulazione, sia per quanto riguarda la gradazione di colore sia per il materiale utilizzato. Quest'ultima componente influisce molto sulla misurazione : in presenza di materiali molto lucidi sono state rilevate cattive riflessioni che hanno compromesso la misura.
La scelta è ricaduta su delle strisce *gialle* di cartoncino da disegno, risultato il meglio rilevabile dal sensore tenute presenti le circostanze

nelle quali si svolge la simulazione, vale a dire un'illuminazione a neon e la traccia nera su un tavolo bianco di materiale lucido.
Una componente altrettanto importante è il corretto posizionamento del marker all'interno del percorso. E' essenziale infatti posizionarlo poco prima della biforcazione, sovrapponendo la "tessera" gialla alla traccia nera del nastro e allineandola al margine destro (vedi Figura 19).

L'algoritmo che permette il riconoscimento dei bivi è riportato di seguito.

```
//ritorna un numero casuale compreso tra 0 e 2 (0 o 1)
turn=Random(2);

if (SV>MK-margine && SV<MK+margine){
//COLORE GIALLO-->MARKER BIVIO TROVATO
   OnFwd(OUT_AC,SpeedFast);
   Wait(twait);
   Off(OUT_AC);
   if (turn==0) {
                              //BIVIO:vai a SINISTRA -->
               //--> gira a sinistra finche non trovi nero

       while (SV<Threshold1 || SV>Threshold2){
                             //finché è diverso da NERO
          OnFwd(OUT_A, SpeedFast);
          OnFwd(OUT_C, SpeedSlow);
          SV = SensorRaw(IN_3);

       }//WHILE DIVERSO DA NERO

   }
   if(turn==1) {
                            //BIVIO : vai a DESTRA!!!  -->
                    //gira a destra finche non trovi nero

       while (SV<Threshold1 || SV>Threshold2){
                                 //finché diverso da NERO
           OnFwd(OUT_C, SpeedFast);
           OnFwd(OUT_A, SpeedSlow);
           SV = SensorRaw(IN_3);

       } //WHILE DIVERSO DA NERO
   }

}//fine if MARKER
```

Il rilevamento di un bivio implica il verificarsi della condizione " `SV > MK-margine && SV < MK+margine` " . La prima istruzione infatti si occupa proprio di tale controllo e in caso affermativo vengono eseguite tutte le istruzioni rimanenti.
La variabile "*SV*" è nota, mentre indichiamo con "*MK*" il valore predefinito (misurato) di intensità luminosa del marker. E' stato misurato un valore pari a "*MK=410*".
A causa della poca precisione del sensore è stato necessario prevedere una tolleranza nella misura. Mediante la variabile "*margine*", si espande leggermente l'intervallo considerato valido per la determinazione del marcatore giallo. Diverse prove hanno portato a stabilire per la suddetta variabile un valore uguale a 10, garantendo cosi una buona precisione.
La scelta della direzione avviene casualmente ed è espressa dalla variabile "*turn*" (turn=0 indica la svolta a sinistra, turn=1 quella a destra).
A seconda della direzione da prendere, il robot dovrà effettuare una rotazione nel verso opportuno, fino a re-agganciarsi alla traccia nera.
Per la svolta a destra, ad esempio, questa condizione è espressa dal ciclo :

```
while (SV<Threshold1 || SV>Threshold2){
            OnFwd(OUT_C, SpeedFast);
            OnFwd(OUT_A, SpeedSlow);
            SV = SensorRaw(IN_3);
        }
```

contenuto nel costrutto *if(turn==1)*. In questo caso mediante le istruzioni "*OnFwd*" vengono settate le velocità dei due motori : il sinistro avrà una velocità superiore rispetto al destro. Infine come ultima istruzione del ciclo si esegue una lettura del valore da parte del sensore in modo tale da poter eseguire nuovamente il controllo sull'allineamento.
Il codice per la svolta a sinistra è analogo a quello appena descritto, naturalmente impostando velocità opposte.

Proprio come per "FollowLine" l'intero codice di riconoscimento dei bivi è posto all'interno di un ciclo *while(true)* in modo tale da terminare il controllo solo in caso di terminazione forzata del programma.

5.2.4 Rilevamento della distanza dagli inseguitori

Il codice di controllo del robot corridore gli consente un'autoregolazione della propria velocità di moto, in base alla distanza dagli inseguitori.
Più precisamente dovrà aumentare la velocità se gli altri robot si avvicinano, diminuirla man mano che si allontanano e fermarsi nel caso non ne rilevi più la presenza.
Le dimensioni ridotte del tavolo a disposizione hanno portato a considerare i robot inseguitori come "*persi*" se la distanza supera i 40cm.
(va precisato che considerare una distanza maggiore avrebbe falsato la simulazione a causa dalle troppo vicine barriere attorno al percorso).
In caso contrario l'andamento del corridore è regolato da funzioni inversamente proporzionali alla distanza misurata.
Per una corretta comprensione vanno specificati i seguenti parametri :

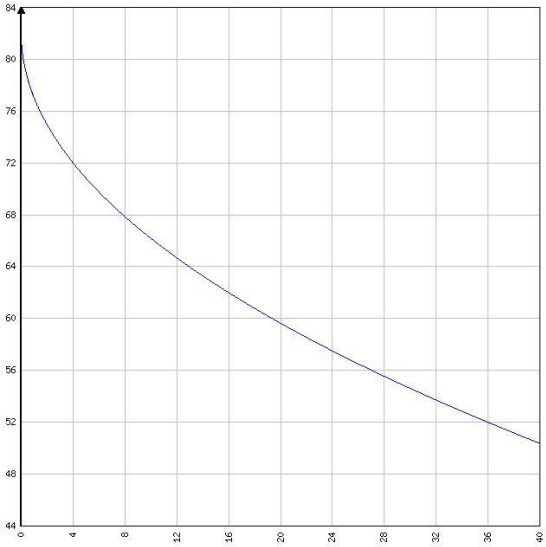
- `dist=SensorUS(IN_4)` : rappresenta il valore letto dal sensore ad ultrasuoni, ovvero la distanza dagli inseguitori salvata nella variabile *dist*.
- `twait` : è espresso in ms e rappresenta il parametro della funzione `Wait(twait)` utilizzata nel programma per sospendere un task per *twait* millisecondi.
- I parametri delle funzioni sono cosi impostati : $mf = 5, qf = 82, ms = 1, qf = 21, mw = 7, qw = 50$

Di seguito il codice che implementa la funzione descritta.

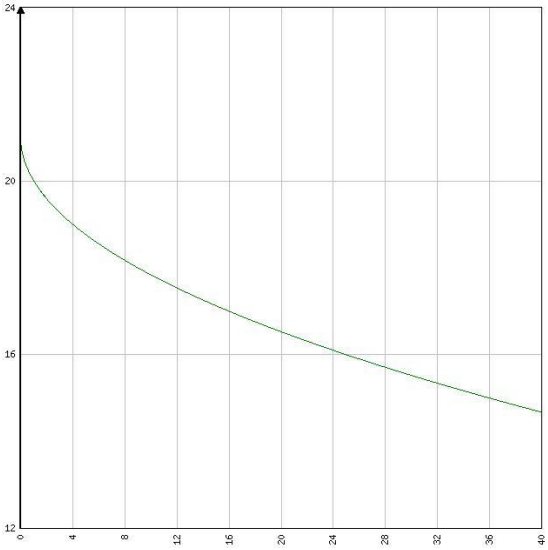
```
dist=SensorUS(IN_4);

while (dist > 40){    //SE PERDE GLI INSEGUITORI
    Off(OUT_AC);
    dist=SensorUS(IN_4);
}
SpeedFast=(-Sqrt(dist)*mf)+qf;       (A)
SpeedSlow=(-Sqrt(dist)*ms)+qs;       (B)
twait=(Sqrt(dist)*mw)+qw;          (C)
```

E' facile osservare che quando la distanza supera i 40cm vengono spenti i motori, il robot si ferma e riprende la sua corsa solo quando gli inseguitori si avvicinano. Tramite le istruzioni (A) e (B) vengono regolati rispettivamente i parametri *SpeedFast* e *SpeedSlow* che controllano il moto del robot. Più precisamente le istruzioni descrivono due funzioni decrescenti con l'aumentare della distanza.

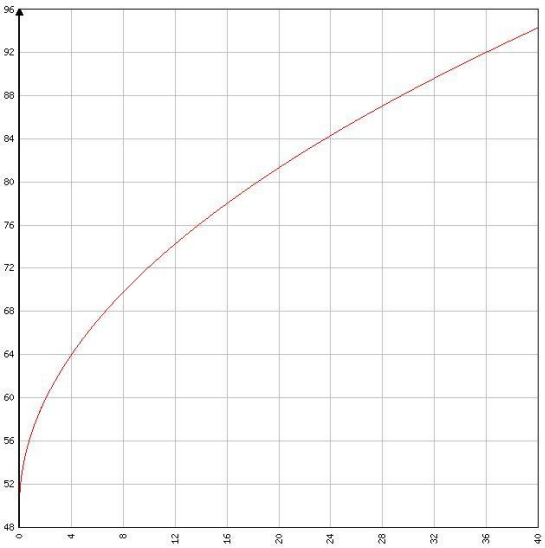
La terza istruzione (C) rappresenta una funzione simile alle precedenti (ma crescente) che si occupa di regolare il parametro twait sopra descritto.
I seguenti grafici e la tabella mostrano l'andamento delle tre funzioni.



(A) – Grafico del parametro SpeedFast in funzione della distanza



(B) – Grafico del parametro SpeedSlow in funzione della distanza



(C) – Grafico del parametro twait in funzione della distanza

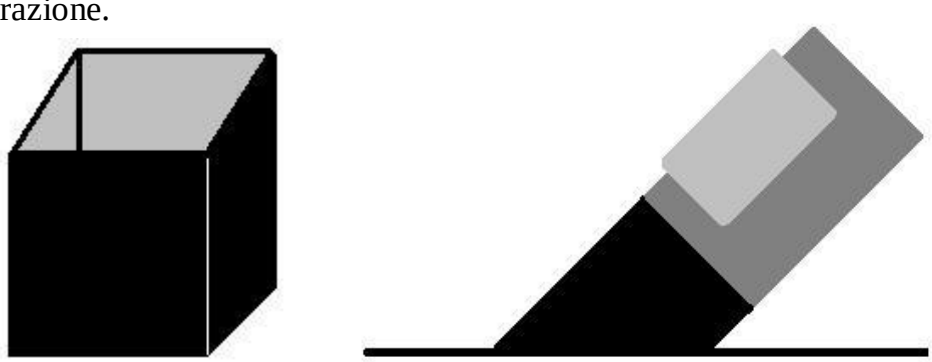
dist	Fast (dist)	Slow (dist)	twait (dist)
0	82	21	50
4	72	19	64
8	68	18	70
12	65	18	74
16	62	17	78
20	60	17	81
24	58	16	84
28	56	16	87
32	54	15	90
36	52	15	92
40	50	15	94

Tabella dei valori assunti dalle tre funzioni

Tali parametri influiscono sull'andamento del robot (vedi 5.2.2 e 5.2.3), permettendogli di godere di un "*feedback*" sullo situazione alle sue spalle.

5.3 Problematiche riscontrate

Durante l'implementazione del programma dedicato al robot corridore si sono manifestate due problematiche principali. Entrambi i problemi sono in gran parte dovuti alla poca precisione e sensibilità del sensore di luce.
Un fattore molto influenzante è l'illuminazione dell'ambiente di svolgimento della simulazione, che ha spesso provocato una cattiva riflessione sul tavolo costituito da materiale lucido. Non potendo variare l'illuminazione è stato tentato di risolvere il problema ricoprendo il piano del tavolo con una tovaglia di carta bianca (opaca). Nonostante siano stati osservati dei miglioramenti rispetto al caso precedente, non risultavano tali da farci considerare a pieno questa opportunità. Abbiamo pertanto deciso di ripristinare il tavolo come in principio e di studiare una soluzione riguardante direttamente il sensore di luce.
Per prima cosa ci siamo assicurati di porre il sensore il più vicino possibile al piano, abbassandolo rispetto alla configurazione iniziale.
La soluzione migliore tuttavia è risultata essere la creazione di un parallelepipedo di cartoncino nero leggermente inclinato su di un lato, posto all'estremità del sensore, in modo tale da prolungare lo stesso fino al piano del tavolo (Figura 26). Cosi facendo si eliminano completamente le rifrazioni provenienti dall'esterno migliorando la flessibilità della misurazione.



b) Vista in prospettiva **a) Vista laterale**

Figura 26 – Parallelepipedo di cartoncino nero

Un secondo problema è stato riscontrato nel mancato riconoscimento del marker, anch'esso dovuto al problema sopra descritto. Per questo motivo è stata prevista l'opportunità di effettuare l'acquisizione del marcatore prima dell'esecuzione del programma. Tuttavia questa soluzione può portare ad avere un valore di marker non sufficientemente distinguibile dagli altri elementi in "gioco". Pertanto si è scelto di non implementare tale funzione per l'utente, ed eventualmente di operare alla modifica direttamente dal codice.

6 Conclusioni

Dalle descrizioni riportate in questo lavoro di tesi sono noti i problemi riscontrati a causa dei limiti tecnici dei robot utilizzati. Considerando inoltre lo spazio a disposizione, si può essere soddisfatti della riuscita di tale simulazione, che rende, in modo sufficientemente corretto, l'idea dell'Encierro di Pamplona.

Una valutazione dell'ambiente di sviluppo consente di affermare che avendo a disposizione uno spazio notevolmente più ampio la simulazione risulterebbe sicuramente più realistica e significativa.

Un utile aspetto sviluppabile in tali condizioni è prevedere delle vere e proprie strade cittadine all'interno del percorso, in modo tale da permettere al robot master di muoversi identificando e scegliendo la via da prendere. Prevedendo inoltre, per il corridore, più di due bivi si aumenterebbe il grado di verosimilità.Infine si potrebbero tranquillamente considerare più robot slave rappresentanti la mandria inseguitrice.

Codice NXC allegato

```
#define Slow 15
#define Fast 50
#define wait 400
#include "BullLib.nxc"

//PARAMETRI CONNESSIONE CON SLAVE
#define BT_CONN 0
#define INBOX 7
#define OUTBOX 3

int SV,MK,NR;
int LoopCount;

sub FollowLine(int loopTime){

  int SpeedSlow=Slow;
  int SpeedFast=Fast;
  int Threshold1=570;
  int Threshold2=590;
  int tempomax=2000;
  int theSpeed;
  string conferma="C";
  int count=0,contadist=0;
  int twait=wait; //tempo di wait trovato bivio
  bool left,sonar=true,sound=false,bt=true;
  int turn,i=0,prova,dist;
  int margine=10; //MARGINE TOLLERANZA MARKER

  //COORDINATE SONAR

  long mf=5,qf=82,ms=1,qs=21;
  long mw=7,qw=0;

  bool toomuchR=false,toomuchL=false;
  int SS=SpeedSlow, SF=SpeedFast;
  long t;

  // imposta tipo e modalità dei sensori
  SetSensorType(IN_3, IN_TYPE_LIGHT_ACTIVE);
  SetSensorMode(IN_3, IN_MODE_RAW);
  SetSensorLowspeed(IN_4);
 if (bt){
    //CONTROLLO LA CONNESSIONE CON LO SLAVE
    BTCheck(BT_CONN);
    while(conferma!="a") {
      ReceiveRemoteString(INBOX,true,conferma);
```

```
    TextOut(0,LCD_LINE6,conferma,false);
    PlayToneEx(550, 500, 1, true);
  }
  MK=410;
    PlayToneEx(210, 500, 4, false);
}  //if (bt)
  while (true)  {
        // legge il valore di luce del sensore
        SV = SensorRaw(IN_3);
        ResetSensor(IN_3);
        NumOut(0, LCD_LINE1,SV,false);
        //ritorna un numero casuale tra 0 e 2 --> 0 o 1
        turn=Random(2);
        count++;
   //COLORE GIALLO -->>> MARKER BIVIO TROVATO
   if (SV>MK-margine && SV<MK+margine){
         OnFwd(OUT_AC,SpeedFast);
         Wait(twait);
         Off(OUT_AC);
         if (turn==0) {
                   //BIVIO  :  vai  a  SINISTRA!!!    -->
            // --> gira a sinistra finche non trovi nero
            if (sound)
              PlayToneEx(440, 500, 2, false);
            TextOut(0,LCD_LINE2,"SINITRA",false);
            Wait(3000);
            while   (SV<Threshold1   ||   SV>Threshold2){
                           //finche' è diverso da NERO
               OnFwd(OUT_A, SpeedFast);
               OnFwd(OUT_C, SpeedSlow);
               SV= SensorRaw(IN_3);
             }//WHILE DIVERSO DA NERO
           }//if (turn==0);
           if(turn==1) {
                      //BIVIO  :  vai  a  DESTRA!!!  -->
               //--> gira a destra finche non trovi nero
               if (sound)
                 PlayToneEx(880, 500, 2, false);
               TextOut(0,LCD_LINE2,"DESTRA",false);
               Wait(3000);
               while  (SV<Threshold1  ||  SV>Threshold2){
                              //finche' diverso da NERO
                  OnFwd(OUT_C, SpeedFast);
                  OnFwd(OUT_A, SpeedSlow);
                  SV= SensorRaw(IN_3);
               } //WHILE DIVERSO DA NERO
            }//if (turn==1)
   }//fine if MARKER
```

```
            // imposta la velocità motore A_RIGHT
            if (SV < Threshold2){
                  OnFwd(OUT_A, SpeedFast);
            }
            else {
                  OnFwd(OUT_A, SpeedSlow);
            }
            // imposta la velocità motore C_LEFT
            if (SV > Threshold1){
              OnFwd(OUT_C, SpeedFast);
            }
            else {
                  OnFwd(OUT_C, SpeedSlow);
            }
      //SONAR
            dist=SensorUS(IN_4);
            ResetSensor(IN_4);
            NumOut(0,LCD_LINE8,dist,false);
       if (sonar){
            while (dist > 40){
                  Off(OUT_AC);            //SE LO PERDE
                  dist=SensorUS(IN_4);
            }
            SpeedFast=(-Sqrt(dist)*mf)+qf;
            SpeedSlow=(-Sqrt(dist)*ms)+qs;
            twait=((Sqrt(dist)*mw)+qw;
       }
       NumOut(0,LCD_LINE4,SpeedFast,false);
       NumOut(0,LCD_LINE5,SpeedSlow,false);
  }//WHILE(TRUE)
  return;
}
task main(){
      string lcStr;
      string svStr;
      string msg;
      // chiamata a subroutine
      FollowLine(10000);
      Wait(10000);
}
```

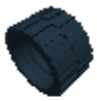
Codice del robot corridore

Istruzioni di montaggio del robot Corridore

Model Name: **corridore**
Number of Bricks: **115**

Step 0

Step 1

1 x 

Step 2

1 x 

1 x 

2 x 

Step 3

2 x 

1 x 

1 x 

Step 4

1 x 

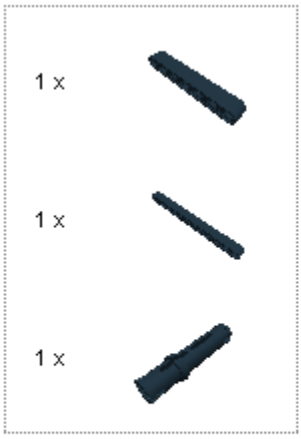
1 x 

2 x 

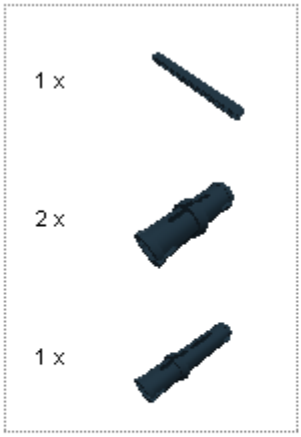
1 x 

3 x 

Step 5



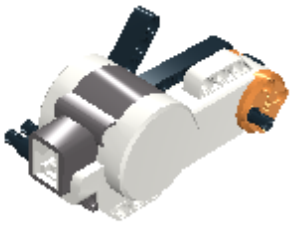
Step 6



Step 7

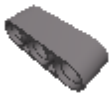
1 x 

3 x 



Step 8

1 x 

1 x 

1 x 

1 x 

Step 9

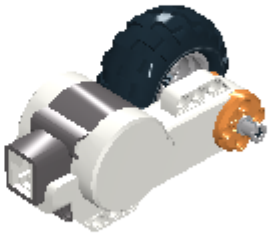
1 x 

1 x 

1 x 

Step 10

1 x 

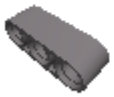


Step 11

1 x 

1 x 

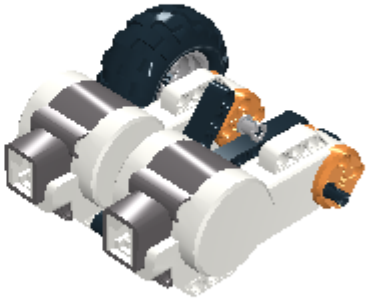
1 x 

1 x 

2 x 

1 x 

1 x 



Step 12

1 x 

Step 13

1 x 

2 x 

1 x 

Step 14

1 x 

1 x 

1 x 

1 x 

Step 15

1 x 

Step 16

1 x 

1 x 

1 x 

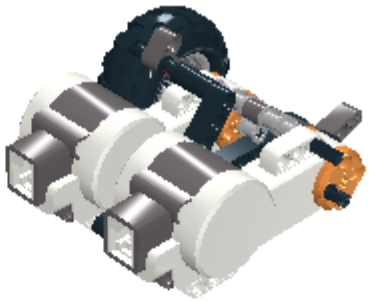
2 x 

2 x 

1 x 

1 x 

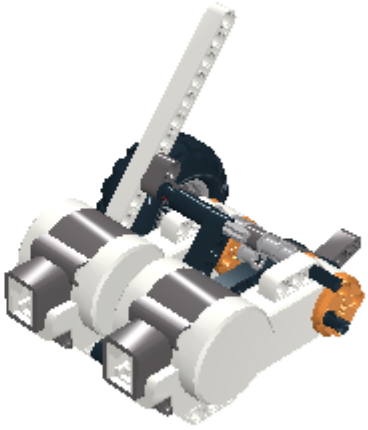
1 x 



Step 17

1 x 

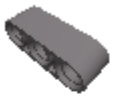
1 x 



Step 18

1 x 

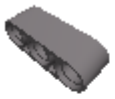
Step 19

1 x 

2 x 

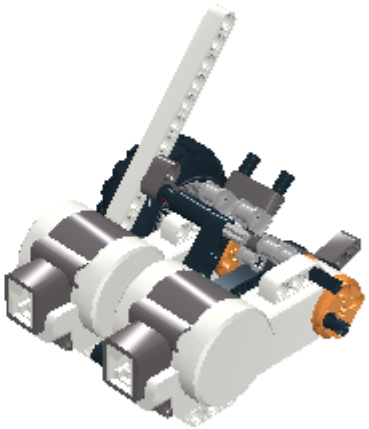
1 x 

Step 20

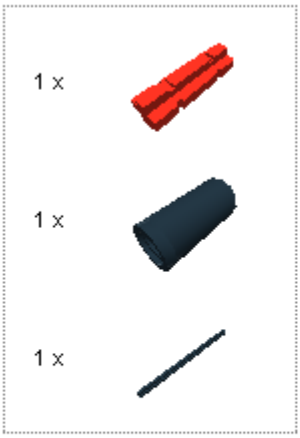
1 x 

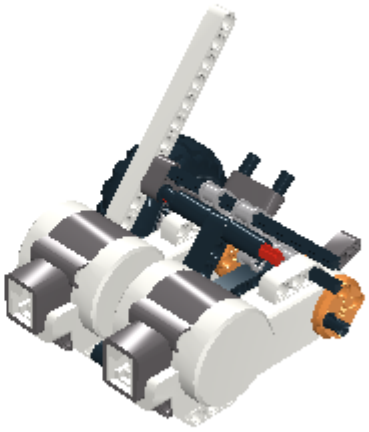
2 x 

2 x 

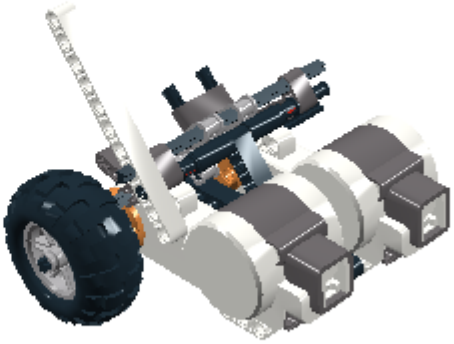


Step 21



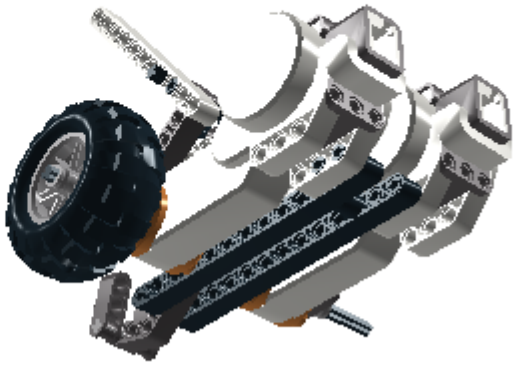


Step 22



Step 23

Step 24



Step 25

Step 26

Step 27

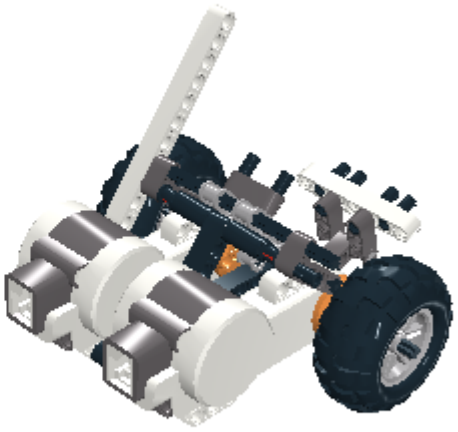
1 x 

1 x 

Step 28

1 x 

1 x 



Step 29

1 x 

Step 30

2 x 

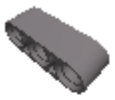
2 x 

Step 31

1 x 

1 x 

Step 32

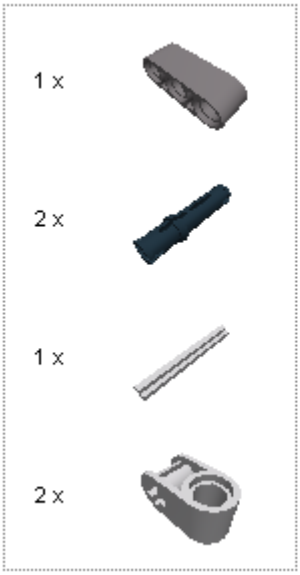
1 x 

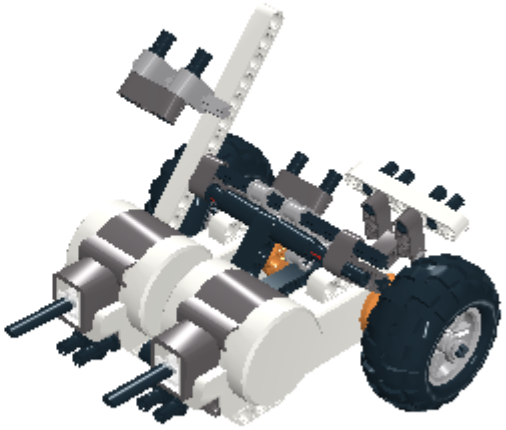
1 x 

1 x 

1 x 

Step 33





Step 34

Step 35

Step 36

Step 37

1 x 

1 x 

Step 38

1 x 

2 x 

1 x 

Step 39

1 x 

1 x 

2 x 

Step 40

1 x 

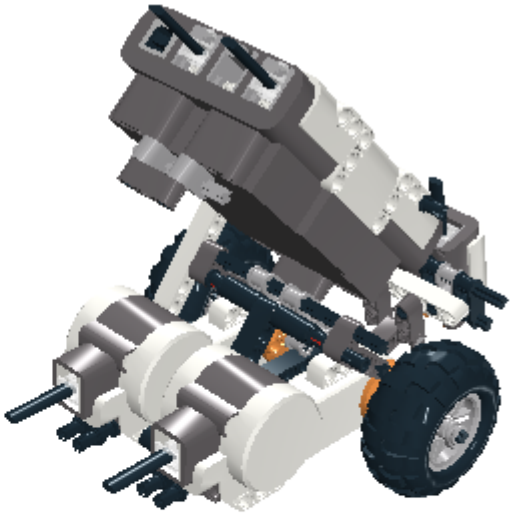
1 x 

2 x 

4 x 

1 x 

1 x 



Step 41

1 x 

1 x 

1 x 

Step 42

2 x 

1 x 

1 x 

Step 43

1 x 

2 x 

1 x 

Step 44

1 x 

1 x 

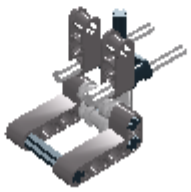
1 x 

1 x 

Step 45

2 x 

2 x 

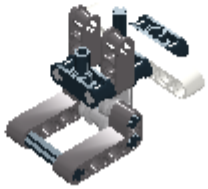


Step 46

1 x 

1 x 

2 x 



Step 47

1 x 

1 x 

1 x 

1 x 

Step 48

1 x 

1 x 

2 x 

6 x 

2 x 

1 x 

1 x 

3 x 

2 x 

2 x 

1 x 

3 x 

2 x 

Step 49

2 x 

2 x 

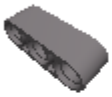
1 x 

3 x 

1 x 

2 x 

1 x 

4 x 

1 x 

2 x 

1 x 

4 x 

1 x 

2 x 

1 x 

10 x 

2 x 

11 x 

3 x 

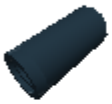
5 x 

3 x 

8 x 

3 x 

10 x 

2 x 

1 x 

2 x 

7 x 

3 x 

4 x 

2 x 

1 x 

2 x 

Bibliografia

L'Encierro

http://www.sanferminencierro.com

http://it.wikipedia.org/wiki/Encierro

San Fermino

http://www.turismo.navarra.es/ita/propuestas/san-fermines/desarrollo/encierro.htm

LEGO Mindstorms

http://mindstorms.lego.com

http://it.wikipedia.org/wiki/LEGO_Mindstorms

http://philohome.com/nxtmotor/nxtmotor.htm

Bricx Command Center

http://bricxcc.sourceforge.net/

NXC Programmer's Guide. J. Hansen, 2007

http://bricxcc.sourceforge.net/nbc