Spazzino Project

Alberto Sarullo
Marco Aspesi

Laboratorio di Intelligenza Artificiale

Università degli Studi di Milano

Anno Accademico 2007/2008

www.aracni.de/spazzinoproject

Indice

Contesto

Il progetto presentato in questo rapporto si è svolto nell'ambito del Laboratorio di Intelligenza Artificiale dell'Università degli Studi di Milano, sotto la direzione del responsabile del Laboratorio stesso, e docente dell'Università, Mario Ornaghi.

Il tema proposto era: "Ideazione e progettazione di un agente intelligente", adottando il modello di "agente intelligente" presentato nel testo "Computational Intelligence" e utilizzando Lego Mindstorm per la costruzione fisica dell'agente.

Spazzino project e' uno dei progetti sviluppati nell'anno accademico 07/08. Si tratta di un agente autonomo intelligente che simula il recupero dei rifiuti di un'area cittadina ottimizzando il tempo totale di uscita del mezzo che copre suddetta area.

Il problema

Scelta un'area cittadina qualsiasi, l'agente autonomo intelligente deve essere in grado di recuperare i rifiuti urbani depositati dai cittadini fuori dalle loro case e negli appositi spazi riservati, ottimizzando il tempo totale di copertura dell'area stessa.

Tale ottimizzazione implica che l'agente non deve solo essere capace di elaborare una possibile strategia per spostarsi tra le varie locazioni di recupero dei rifiuti, ma deve *intelligentemente* elaborare la migliore.

L'Agente

L'agente autonomo intelligente di cui si tratterà da qui in avanti, come si evince dal problema esposto al paragrafo predente, deve essere in grado di:

- recuperare e immagazzinare i rifiuti urbani dell'area cittadina di interesse;
- muoversi tra le varie locazioni di recupero dell'area cittadina di interesse;
- elaborare una strategia o piano che gli permetta di coprire l'area cittadina di interesse nella maniera migliore.

La natura delle capacità di cui deve essere munito l'agente sono dunque differenti: le prime due sono funzioni che l'agente deve svolgere, l'ultima implica invece un ragionamento.

Per tale motivo si è deciso di dotare l'agente di moduli differenti, adibiti a compiti specifici, che interagiscono fra loro:

1. un sistema esperto, il pianificatore, che calcola il piano per il recupero dei rifiuti attraverso l'uso di procedure di *inferenza* adeguate.

   Il piano non sarà altro che una sequenza di azioni che garantiranno il raggiungimento dell'obiettivo desiderato.
2. un robot, costruito con tecnologia Lego Mindstorm, che esegue il piano e registra i tempi di spostamento tra le locazioni di recupero dell'area cittadina di interesse;

3. un'interfaccia che funge da interprete bidirezionale tra i due differenti linguaggi dei moduli precedentemente presentati: il linguaggio formale del pianificatore e il linguaggio ad alto livello del robot.

   Bidirezionale perché lato robot traduce la sequenza di azioni logiche prodotte dal pianificatore in una sequenza di azioni ad alto livello eseguibili dal NXT Intelligent Brick del Lego Mindstorm. Lato pianificatore, invece, consente al robot di trasmettere le informazioni acquisite alla base di conoscenza di interesse del piano;

4. ultimo, ma non di irrilevante importanza, l'ambiente di simulazione dell'area cittadina di interesse su cui operano i primi due moduli, il pianificatore e il robot.

   L'agente, dunque, opera sotto l'ipotesi di conoscenza delle posizioni delle locazioni di recupero dei rifiuti dell'area cittadina e dei collegamenti tra esse.

L'ottimizzazione del piano avviene grazie all'esplorazione dell'area cittadina da parte del robot che immagazzina i dati utili al pianificatore, i *tempi effettivi* di spostamento tra le locazioni di recupero, per migliorare il suo tempo di uscita.

Al termine di ogni esecuzione del piano, infatti, quest'ultimo viene ricalcolato con i nuovi dati acquisiti per aumentarne l'efficienza e via via cercare di raggiungere il piano migliore.

Presentata la struttura modulare dell'agente, entriamo ora nel dettaglio dei singoli moduli.

Modulo 1: il pianificatore

Un pianificatore automatico è un agente intelligente che opera in un certo dominio e che date:

- una rappresentazione dello stato iniziale
- una rappresentazione del goal
- una descrizione formale delle azioni eseguibili

sintetizza dinamicamente il piano di azioni necessario per raggiungere il goal.

Dalla definizione risulta evidente che per risolvere un problema di pianificazione è necessario procedere in 3 fasi sequenziali:

- l'***individuazione degli elementi fondamentali*** che costituiscono il problema di pianificazione:

  il ***dominio***: l'insieme dei predicati che definiscono le entità logiche che costituiscono il modello del sistema su cui opera il pianificatore;

  lo ***stato iniziale***: è la configurazione di partenza in cui si trovano le entità logiche che costituiscono il dominio del problema;

  il ***goal***: obiettivo/i che deve raggiungere il piano;

  le ***azioni***: azioni eseguibili sul dominio, caratterizzate da precondizioni e postcondizioni, che comportano un'evoluzione dello stato del mondo su cui si opera.

  Le precondizioni rappresentano le condizioni che devono essere verificate affinché l'azione possa essere eseguita; le postcondizioni invece rappresentano gli effetti dell'azione stessa sul mondo;
- la ***scelta di un linguaggio formale*** che permetta di rappresentare gli elementi del problema in modo adeguato;
- la ***modellazione*** degli elementi del problema con il linguaggio scelto.

Effettuati questi passi, un possibile approccio per il raggiungimento della soluzione del problema è quello di tradurre il modello ottenuto dall'applicazione delle fasi precedenti, in un programma logico risolvibile da un sistema già presente in letteratura (nel caso in esame, Smodels [3][4]).

Il modo di operare appena descritto è quello che si è utilizzato in Spazzino Project.

Vediamo dunque fase per fase come si è proceduto.

Fase 1: l'individuazione degli elementi del problema di pianificazione

Il problema di pianificazione affrontato in Spazzino Project è la simulazione del recupero dei rifiuti di un'area cittadina con relativa ottimizzazione del tempo totale di uscita del mezzo che copre suddetta area.

Gli elementi fondamentali che lo caratterizzano sono:

- l'area cittadina, il ***dominio***, modellabile con un grafo non orientato.

  Si può pensare infatti di rappresentare tale area con una coppia <S,A>, in cui l'insieme dei nodi S rappresenta tutte le locazioni di recupero dei rifiuti e l'insieme A degli archi pesati i collegamenti tra le varie locazioni.

  Gli archi sono pesati per tenere traccia del tempo che impiega il robot a percorrere le strade cittadine che gli archi stessi modellano;

- la locazione di partenza del mezzo e le locazioni dei rifiuti, lo ***stato iniziale***;

- il recupero dei rifiuti, il ***goal*** del piano.

  Per ottenere l'obiettivo si deve effettuare una visita completa del grafo, ogni nodo deve essere esplorato.

  Allo stesso tempo, per garantire l'ottimizzazione del tempo totale di uscita del mezzo che copre l'area cittadina, si è previsto che ogni nodo sia raggiunto una sola volta dal mezzo nel piano;

- lo spostamento del mezzo nell'area cittadina e il recupero dei rifiuti nelle varie locazioni di recupero: le ***azioni*** da eseguire.

Fase 2: la scelta del linguaggio formale

Individuati gli elementi fondamentali che descrivono il problema di pianificazione di interesse, si è concentrata l'attenzione sulla scelta di un opportuno linguaggio formale che permettesse di rappresentare il problema in modo adeguato.

Nel caso di Spazzino Project, tale scelta è caduta sul ***linguaggio Ak*** per 2 motivi:

- Ak è un linguaggio formale, derivato dal linguaggio A [1], che consente di specificare problemi di pianificazione in cui l'agente deve acquisire informazioni dal mondo esterno per decidere quali azioni eseguire.

  In Spazzino Project, anche se queste problematiche non sono state affrontate per mancanza di tempo (vedere la sezione sviluppi futuri a riguardo), l'agente potrebbe necessitare di informazioni dall'ambiente di interesse.

  Ad esempio, se il contenitore dei rifiuti di cui è dotato il mezzo di recupero avesse una capacità limitata rispetto all'area da coprire, il piano dovrebbe controllare la capienza rimanente del contenitore prima di eseguire l'azione di recupero;

- Ak è traducibile in modo automatico in un programma logico risolvibile da un sistema già presente in letteratura, Smodels [3][4], in grado di generare, a seconda delle regole di inferenza definite, piani condizionali[1] o conformanti[2].

  L'interprete in grado di effettuare tale conversione è scritto in Prolog[3] e si chiama tr.pl.

Definito il linguaggio di rappresentazione del problema vediamo come il problema stesso è stato modellato.

Fase 3: la modellazione del problema di pianificazione

La terza, e ultima fase di rappresentazione del problema di pianificazione di interesse, prevede la modellazione di tutti gli elementi individuati nella prima fase.

In questo paragrafo però, per non distrarre il lettore con righe di codice che rendono la comprensione difficile, ci limitiamo a illustrare i costrutti del linguaggio Ak con espliciti riferimenti al problema che abbiamo affrontato. Il modello è visionabile nella sua totalità nel paragrafo successivo.

Prima di entrare nel merito della modellazione Ak, si presentano di seguito limitazioni di carattere hardware che si riflettono nella definizione del problema:

- le locazioni di recupero e i contenitori dei rifiuti sono in numero limitato: 5 locazioni e 3 contenitori.
  station(l) :- member(l,[1,2,3,4,5]).

  garbage_container(l) :- member(l,[1,2,3]).

- non tutte le locazioni di recupero sono connesse in modo diretto; il grafo che modella l'area cittadina dunque non è completo[4].

  a(1,2,q) ***a(3,4,x)***

  a(1,5,t) ***a(3,5,y)***

  a(2,3,u) ***a(4,5,z)***

  a(2,4,v) *P={q,t,u,v,x,y,z}, insieme dei pesi degli archi*

Per ulteriori chiarimenti consultare il paragrafo modulo 4: l'ambiente.

1 Piano condizionale: un piano che contiene azioni di “sensing” e condizioni di “sensing” (strutture del tipo“if-then-else”).
2 Piano conformante: un piano specificato da una sequenza di azioni che conducono al goal indipendentemente dal valore dei fluenti sconosciuti nello stato iniziale dove per fluente si intende un aspetto parziale del dominio che varia nel tempo.
3 Prolog: ***PRO****gramming in* ***LOG****ic*) è un linguaggio di programmazione che consente l'espressione di un problema in forma logica. Riferimento: http:\\www.swi-prolog.org
4 Grafo completo: dato il grafo <G,E>, il grafo si definisce completo se ogni suo vertice x ϵ G è collegato a tutti i vertici rimanenti in {G - x}

I costrutti del linguaggio Ak per la rappresentazione di problemi di pianificazione sono i seguenti:

- ***causes(a,f,{p_1,...,p_n})*** : per rappresentare gli effetti di un'azione.
  Ad esempio causes(go(I,J),robot_is(J),[]) modella l'effetto dell'azione di spostamento del robot dalla locazione I alla locazione J. Rende vero il fatto che il robot si trovi nella locazione J;

- ***executable(a,{ p_1,...,p_n})*** : per rappresentare le precondizioni di un azione.
  Ad esempio executable(go(I,J),[robot_is(I), fine_carico(I), neg(visited(J))]) modella il fatto che il robot possa spostarsi dalla locazione I alla locazione J se e solo se si trova in I, i rifiuti che si trovano nella locazione I sono stati recuperati e la locazione J non è ancora stata visitata;

- ***caused(f,{p_1,...,p_n})*** : per rappresentare *leggi di causalità statica.*
  Ad esempio caused([in(I,R)],neg(in(J,R))) *modella la semplice legge causale che se il conteiner dei rifiuti R è in I non può trovarsi allo stesso tempo in J*;

- ***determines(a,{l_1,...,l_n})*** : per rappresentare gli effetti di un'azione di sensing;

- ***initially(f)*** : per rappresentare lo stato iniziale.
  Ad esempio initially(robot_is(1)) modella che nello stato iniziale il robot si trova alla locazione 1;

dove *f, {p_1,...,p_n} e {l_1,...,l_n}* sono fluenti[5] e *a* è un'azione.

5 Fluente: aspetto parziale del dominio la cui verità varia nel tempo. Storicamente il termine è stato introdotto da Newton per indicare funzioni sul dominio del tempo. Newton definisce la *flussione*, la derivata di un fluente.

Il codice del pianificatore

Predicati di dominio per l'instanziazione

```
garbage_conteiner(I) :- member(I,[1,2,3]).
station(I) :- member(I,[1,2,3,4,5]).

%%Grafo non orientato <S,A> con P={q,t,u,v,x,y,z} e P ϵ N, insieme dei pesi degli archi
a(1,2,q).
a(1,5,t).
a(2,3,u).
a(2,4,v).
a(3,4,x).
a(3,5,y).
a(4,5,z).

%%I e J sono 2 stazioni differenti
different_stations(I,J) :- station(I), station(J), I\=J.

%%Esiste il cammino I->J o J->I
arc(I,J) :- a(I,J,_P);a(J,I,_P).
```

I fluenti primitivi

```
%% i rifiuti del container R sono in attesa di essere recuperati nella stazione I
fluent(in(I,R)) :- station(I), garbage_container(R).

%% il robot e' nella stazione I
fluent(robot_is(I)) :- station(I).

%% i rifiuti del container R sono stati recuperati
fluent(recovered(R)) :- garbage_container(R).

%%la stazione J è stata visitata
fluent(visited(J)) :- station(J).
```

Le azioni

```
action(go(I,J)) :- different_stations(I,J).
action(takeon(R,I)) :- garbage_container(R), station(I).
```

Causalità statica

%%causa neg(in(I,R)): il container R non può essere alla stazione I e J contemporaneamente

```
caused([in(I,R)],neg(in(J,R))) :- different_stations(I,J), garbage_container(R).
```

%%causa neg(in(I,R)): un container R recuperato non è più nella stazione I in attesa di essere caricato

```
caused([recovered(R)],neg(in(I,R))) :- station(I), garbage_container(R).
```

%%causa neg(robot_is(I)): il robot non può essere nella stazione I e J contemporaneamente

```
caused([robot_is(I)],neg(robot_is(J))) :- different_stations(I,J).
```

%%causa neg(recovered(R)): il container R non e' stato recuperato

```
caused([in(I,R)],neg(recovered(R))) :- station(I), garbage_container(R).
```

%%il container R non è nella stazione I

```
caused([recovered(R)],fine_carico(I)) :- station(I), garbage_container(R).
caused(L,fine_carico(I)) :- station(I), setof(neg(in(I,R)), garbage_container(R),L).
```

%%la stazione J non è ancora stata visitata

```
caused([robot_is(I),in(J,R)],neg(visited(J))) :- different_stations(I,J), garbage_container(R).
```

Precondizioni

%%il robot recupera i rifiuti dal container R

```
executable(takeon(R,I),[robot_is(I),in(I,R)]) :- garbage_container(R), station(I).
```

%%il robot va da I a J

```
executable(go(I,J),[robot_is(I), fine_carico(I), neg(visited(J))]) :- arc(I,J).
```

Causalità dinamica

```
%%cosa rende vero e cosa rende falso l'azione go
causes(go(I,J),robot_is(J),[]) :- arc(I,J).
causes(go(I,J),visited(J),[]) :- arc(I,J).
causes(go(I,J),neg(robot_is(I)),[]) :- arc(I,J).

%%cosa rende vero e cosa rende falso l'azione takeon
causes(takeon(R,I),recovered(R),[]) :- garbage_container(R), station(I).
causes(takeon(R,I),neg(in(I,R)),[]) :- garbage_container(R), station(I).

%% SENSING (determines) : codice aggiungibile
```

Stato Iniziale

```
%% robot in stazione 1
initially(robot_is(1)).

%%container 1 in 1 , container 2 in 3 , container 3 in 4
initially(in(1,R)) :- garbage_container(R),R=1.
initially(in(3,R)) :- garbage_container(R),R=2.
initially(in(4,R)) :- garbage_container(R),R=3.
```

Goals

```
sgoal(recovered(1)).
sgoal(recovered(3)).
sgoal(recovered(4)).
```

Modulo 2: il robot

Il robot è il modulo hardware di Spazzino Project. Il robot svolge due compiti:

1. esegue il piano di cui si è trattato nel paragrafo precedente;
2. registra i tempi di spostamento tra le locazioni di recupero dell'area cittadina di interesse di cui si tratterà nella sezione modulo 4;

Il robot è stato progettato in maniera da essere modulare e facilmente aggiornabile, compatibilmente con i limiti imposti dai mattoncini Lego.

La scelta di dedicare molto tempo alla progettazione è stata dettata dalla volontà di affrontare già in fase di progettazione una serie di problemi comuni alle costruzioni Lego, garantendo sufficiente flessibilità per correggere quei problemi affrontabili solo a costruzione del robot ultimata.

Per garantire al robot una traiettoria controllata e prevedibile, è stata posta grande attenzione all'equilibratura dello stesso.

Il risultato è che pur avendo una struttura fortemente asimmetrica, il baricentro del robot è perfettamente centrato: più del 95% del peso del robot si scarica sulle ruote motrici, e meno del 5% sulla ruota anteriore, il cui unico compito è quello di mantenere costante la distanza tra i sensori di luminosità e il pavimento.

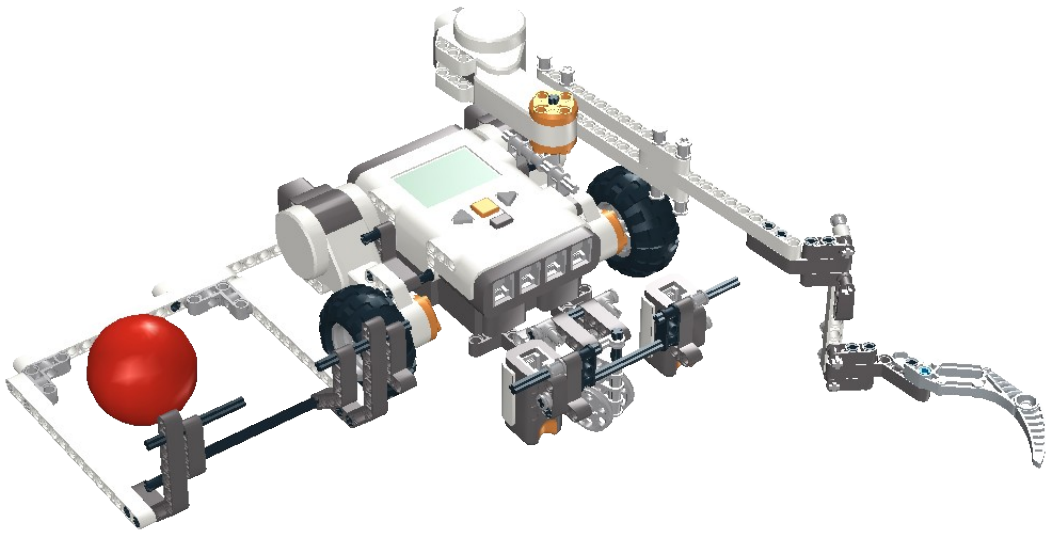


Illustrazione 1: Vista generale del robot

L'architettura fisica del robot è organizzata in moduli facilmente riposizionabili, tutti agganciati in modo diretto all'unità centrale:

- NXT Intelligent Brick
- Sensori luminosita (dx, sx)
- Motori e ruote motrici (dx, sx)
- Ruota anteriore
- Motore e braccio raccattapalle
- Contenitore palline

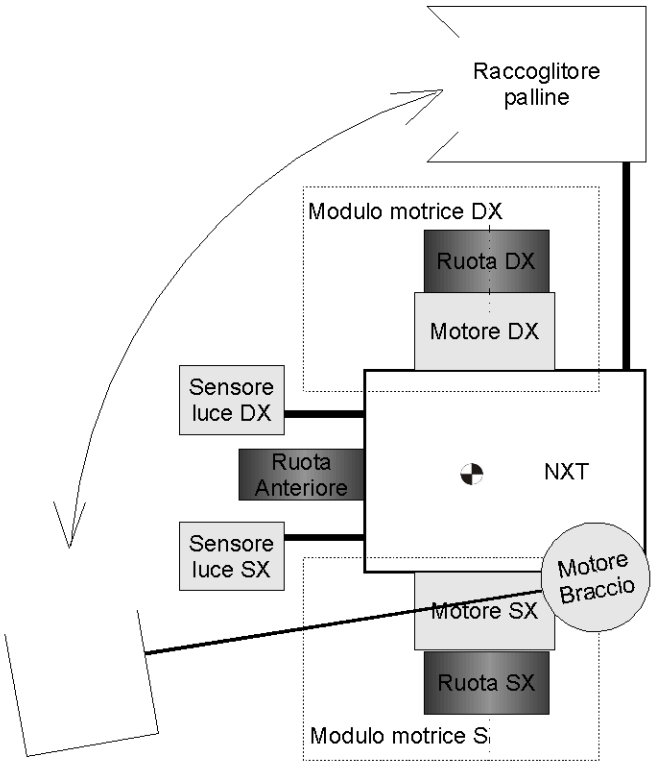


Illustrazione 2: Architettura modulare del robot

NXT Intelligent Brick

Si tratta del modulo più pesante, che contiene le batterie ed i circuiti necessari al funzionamento del robot.

E' stato posizionato al centro del robot per garantire al mezzo una costruzione equilibrata e una solidità non ottenibile altrimenti.

Tutti i moduli di cui è costituito il robot sono fisicamente agganciati al NXT per evitare problemi di interdipendenza tra gli stessi: è possibile spostare buona parte dei moduli senza dover modificare la struttura del robot.

Sensori luminosità

Il robot è dotato di 2 sensori di luminosità, posizionati anteriormente, responsabili della lettura del valore di luminosità del terreno su cui è basato il controllo dello spostamento da una locazione ad un'altra del mezzo.

I sensori di luminosità sono tenuti ad una distanza costante dal terreno dalla ruota anteriore.

Per limitare le interferenze tra la luce ambientale e le letture dei sensori, i sensori di luminosità sono circondati da un paraluce realizzato ad hoc in cartoncino.

Motori e ruote motrici

Le 2 ruote motrici sono responsabili dei movimenti rettilinei e rotatori del robot. Sulle ruote motrici viene scaricato il 95% del peso del robot, e per questa ragione è stata posta grande attenzione alla loro resistenza meccanica.

E' possibile spostare i moduli in avanti e indietro per modificare l'equilibratura del robot.

Ruota anteriore

La ruota anteriore ha la funzione di sostenere il peso del braccio raccattapalle, senza intralciare i movimenti traslato-rotatori del robot.

Il robot è stato costruito in maniera da scaricare il proprio peso quasi interamente sulle ruote motrici, evitando di sovraccaricare la ruota anteriore di pesi che intralcerebbero il suo libero movimento.

L'altezza e l'avanzamento della ruota anteriore è modificabile di 1 cm, per permettere eventuali aggiustamenti fini del movimento del robot.

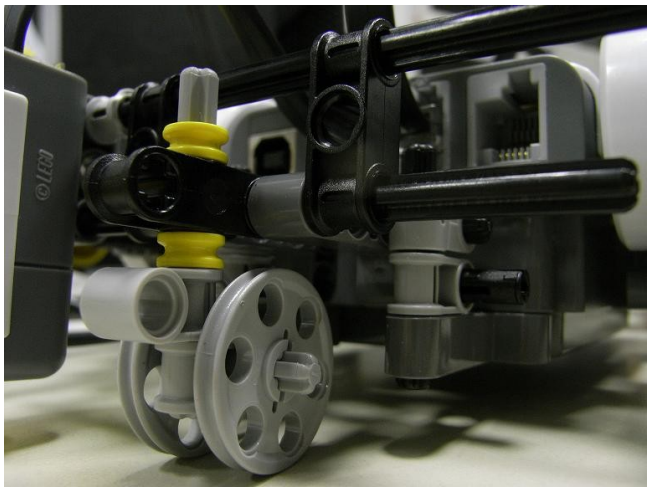


Illustrazione 3: Dettaglio del ruotino anteriore

Motore e braccio raccattapalle

Il braccio raccattapalle è responsabile della cattura delle palline.

E' stato realizzato in modo tale che il motore faccia da contrappeso al peso del

braccio stesso, garantendone l'equilibrio meccanico, ovvero poche vibrazioni scaricate sull'unità centrale quando il braccio viene messo in movimento.

Il braccio raccattapalle è inoltre responsabile dell'apertura della porta a senso unico del contenitore delle palline.

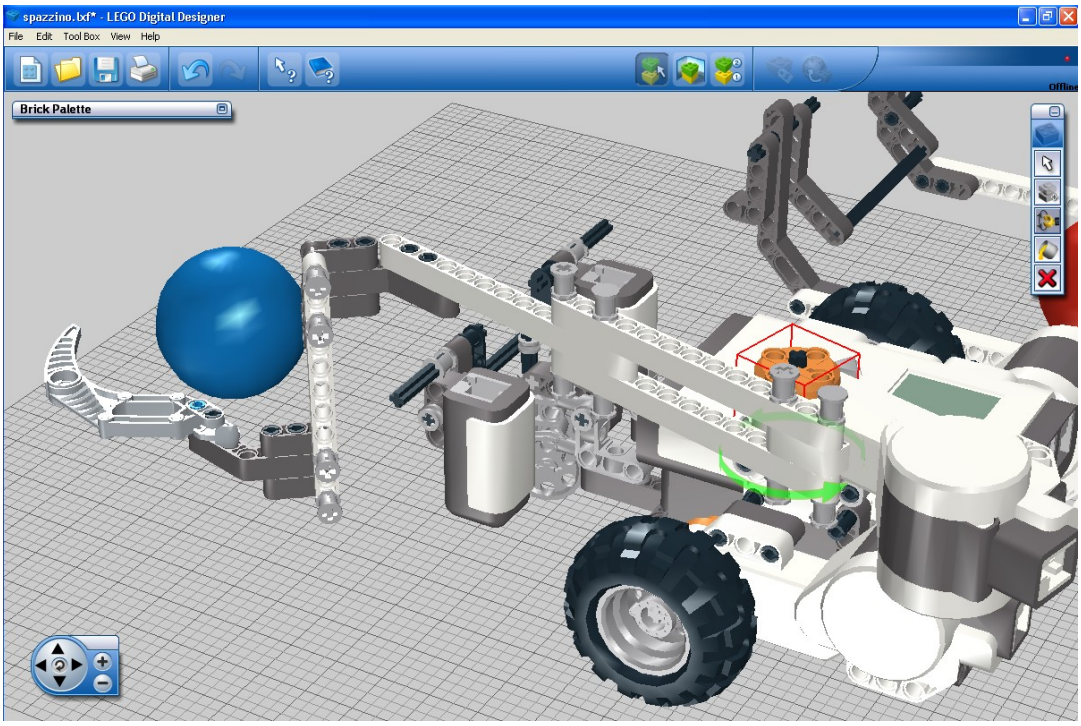


Illustrazione 4: Il braccio raccattapelle - Lego Digital Designer

Contenitore palline

Il contenitore per le palline è posizionato sul lato destro del robot, in maniera da bilanciare il peso del braccio raccattapalle.

Le palline sono appoggiate per terra, per non scaricare sul robot il proprio peso; l'attrito volvente delle palline, quando il robot è in marcia, è trascurabile e non intralcia la traiettoria lineare del robot.

Per favorire l'ingresso delle palline nel contenitore evitandone la fuoriuscita, è stata costruita una porta a senso unico, che viene aperta poco prima che ciascuna pallina venga inserita nel braccio, e chiusa ad ingresso ultimato.

Per aprire e chiudere la porta, si è sfruttato il movimento del braccio raccattapalle.

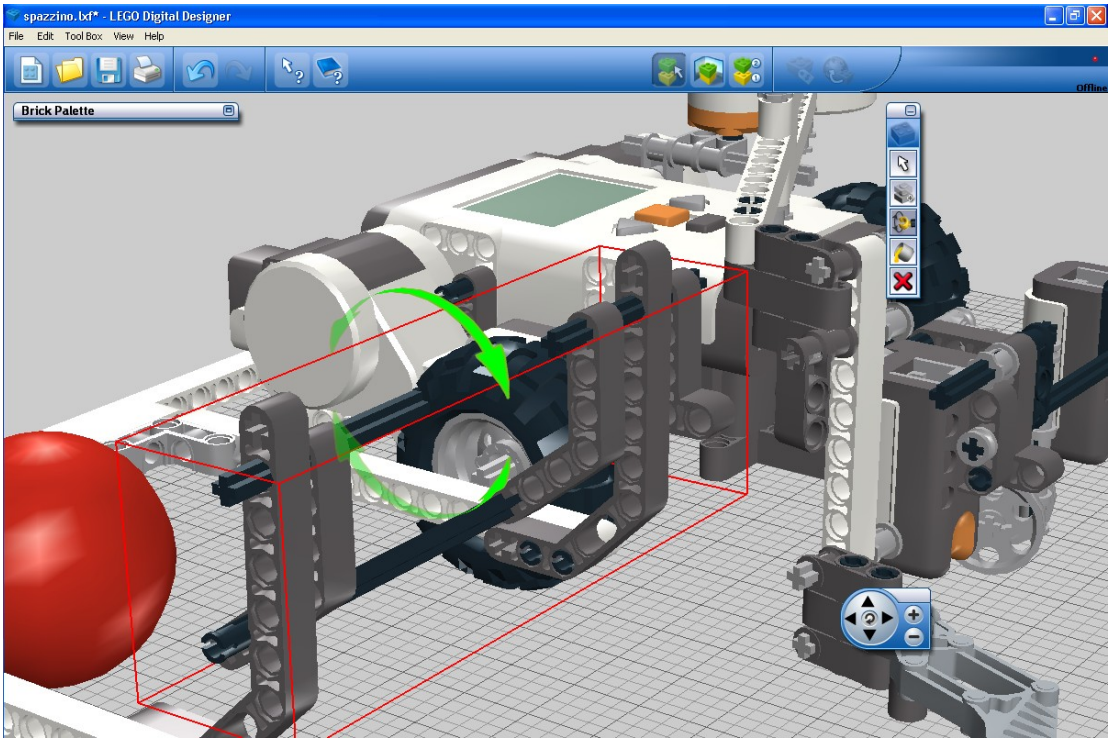


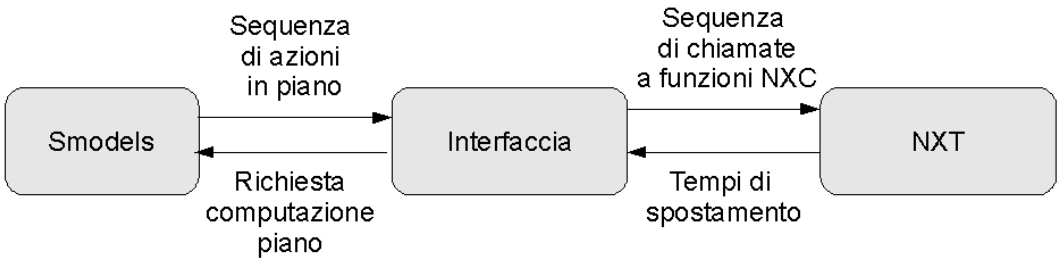
Illustrazione 5: Sistema di apertura contenitore palline - Lego Digital Designer

Modulo 3: l'interfaccia

La necessità di creare un interprete bidirezionale tra il linguaggio formale del pianificatore e il linguaggio ad alto livello del robot, ha convinto a ideare e realizzare un'interfaccia software che svolga i seguenti compiti:

- esecuzione parametrica del modello formale, visto nel modulo 1, tramite chiamata al sistema Smodels;
- traduzione del piano di soluzione del problema di pianificazione in una sequenza di azioni eseguibili dal robot.

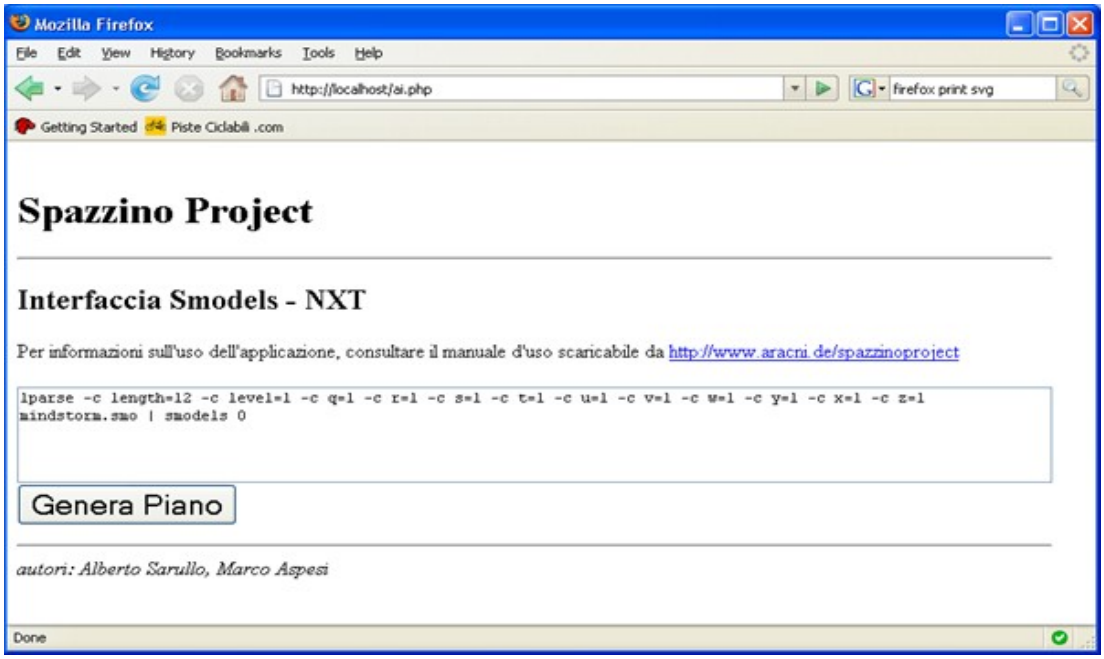
Il pianificatore automatico ideato, infatti, prevede che Smodels, dotato del modello formale del problema, generi il piano di soluzione che il robot deve eseguire. Effettuato il piano, poi, il robot stesso restituisce i dati necessari a Smodels per migliorare il piano precedentemente calcolato. Il linguaggio di Smodels però non è comprensibile dal robot e viceversa. Da qui l'esigenza di creare un layer software che faccia da tramite tra i due sistemi.



Per realizzare il layer abbiamo creato una applicazione Php dotata di interfaccia web: si passano all'applicazione i tempi effettivi di spostamento tra le locazioni di recupero registrate dal robot (alla prima esecuzione, come si vede in figura, tutti inizializzati al valore 1), si preme il pulsante “Genera Piano”, e si ottengono direttamente le istruzioni d'azione da inviare al robot stesso.

La scelta di realizzare un'applicazione web è stata dettata dalla volontà di rendere l'interfaccia raggiungibile ovunque, pur di avere un computer connesso ad internet.

La scelta del Php, quale linguaggio di programmazione per la realizzazione dell'interfaccia, è stata invece dettata dal fatto che tale linguaggio è cross-platform e i suoi tempi di sviluppo sono estremamente rapidi.

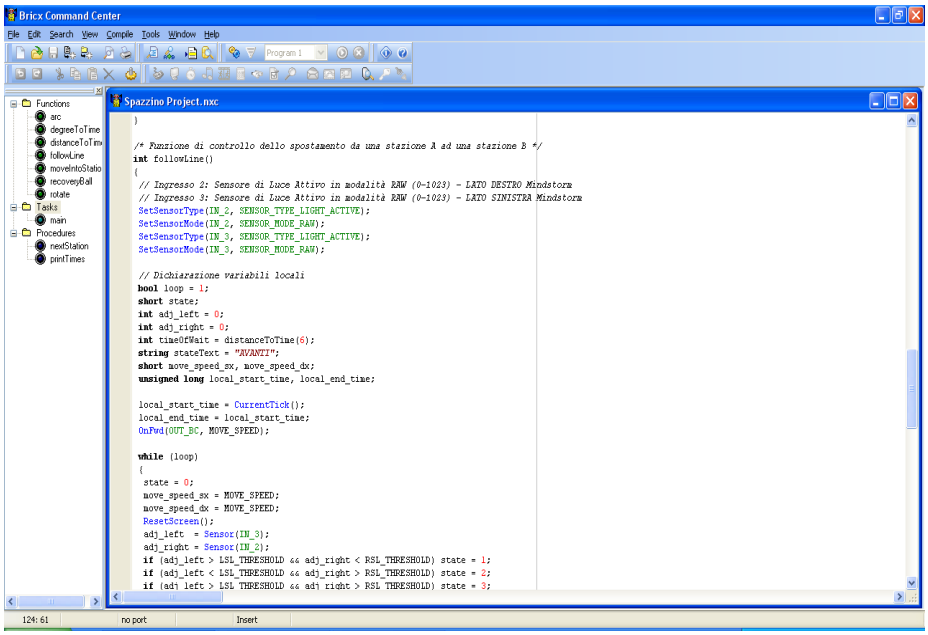

Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/ai.php
firefox print svg
Getting Started Piste Ciclabili .com
Spazzino Project
Interfaccia Smodels - NXT
Per informazioni sull'uso dell'applicazione, consultare il manuale d'uso scaricabile da http://www.aracni.de/spazzinoproject
lparse -c length=12 -c level=1 -c q=1 -c r=1 -c s=1 -c t=1 -c u=1 -c v=1 -c w=1 -c y=1 -c x=1 -c z=1
mindstorm.smo | smodels 0
Genera Piano
autori: Alberto Sarullo, Marco Aspesi
Done

Nella descrizione dell'interfaccia si è illustrato come il piano venga generato e tradotto in azioni interpretabili dal robot. Non si è però chiarito come tali azioni siano inviate al robot e sulla base di quali conoscenze il robot le esegua.

Il motivo è che l'interfaccia, lato robot, si occupa solamente di estrarre, secondo il criterio di ottimo definito dal problema, la migliore delle soluzioni restituite da Smodels, e di convertire tale piano in una sequenza di chiamate a funzione interpretabili da un altro modulo aggiuntivo finora mai citato e che qui si introduce: il Bricx Command Center.

Bricx Command Center

Il Bricx Command Center è un ambiente di sviluppo standard per la programmazione del NXT Intelligent Brick che comunica con il Lego Mindstorm tramite cavo USB, preferibile, o protocollo Bluetooth.



E' un ambiente di facile utilizzo che consente di scrivere codice in diversi linguaggi di programmazione eseguibili su Lego Mindstorm.

Tra i vari linguaggi si è scelto di preferire NXC, Not eXactly C, per diversi motivi:

- è un linguaggio di alto livello, a differenza ad esempio di un altro linguaggio implementabile in Bricx, l'NBC o Next Byte Code che è del tutto simile al linguaggio macchina (assembler);
- è simile al C, un linguaggio molto diffuso e quindi di più facile comprensione e soprattutto molto flessibile per la sua modularità;
- è dotato di funzioni standard affidabili per il controllo dei motori e dei sensori di cui è dotato il robot presentato nel paragrafo dal titolo modulo 2.

Di seguito si illustrano le chiamate a funzione generate dall'interfaccia di Spazzino

Project mostrando a quali azioni del piano esse fanno riferimento:

- *nextStation(30, ballsNumber)* e *times[1] = followLine()* corrispondono all'azione del piano go(I,J) e permettono al robot di spostarsi dalla locazione I alla locazione J dell'area cittadina, registrando opportunamente in un array il tempo di spostamento da I a J.

  Gli angoli di rotazione del robot per muoversi da una stazione all'altra sono fissi e conosciuti dall'interfaccia di Spazzino Project. Si modella il fatto che l'ingresso di una strada cittadina non sia mutevole nel tempo. Una modifica dell'area cittadina richiede un aggiornamento di queste informazioni.

- *ballsNumber = recoveryBall(ballsNumber)* corrisponde all'azione del piano takeon(R,I) e permette al robot di recuperare i rifiuti I presenti nella stazione R, aumentando il numero complessivo di container di rifiuti recuperati. Questo valore è rilevante perché maggior carico significa più attrito e quindi maggiori difficoltà negli spostamenti per il robot.

Il codice del robot

```
#include "NXCDefs.h"
```

Dichiarazione delle costanti

```
#define MOVE_SPEED 75
#define MOVE_ARM 75
#define LSL_THRESHOLD 320
#define RSL_THRESHOLD 320
#define STATION_RADIUS 17
#define ARRAY_LENGTH 10
```

Dichiarazione delle funzioni

```
/* Converte dall'unità di misura centimetro all'unità di misura millesimo di secondo  */
int distanceToTime(int distance){
  return ((distance*1000)/(-3 + 10*MOVE_SPEED/25));
}

/* Converte dall'unità di misura grado all'unità di misura millesimo di secondo  */
int degreeToTime(int degree){
  return ((abs(degree)*1000)/((MOVE_SPEED/25 -1)*67+39));
}

/* Ruota il Mindstorm di degree gradi */
int rotate(int degree, int balls){
  int timeOfWait = (degreeToTime(degree) + balls*15);
  if (degree > 0){
    OnRev(OUT_B, MOVE_SPEED);
    OnFwd(OUT_C, MOVE_SPEED);
  }else{
    OnRev(OUT_C, MOVE_SPEED);
    OnFwd(OUT_B, MOVE_SPEED);
  }
  Wait(timeOfWait);
  Off(OUT_BC);
}
```

```
/* Muove il mindstorm al centro della stazione di recupero */
int moveIntoStation(int distance){
  int timeOfWait = distanceToTime(distance);
  OnFwd(OUT_BC, MOVE_SPEED);
  Wait(timeOfWait);
  Off(OUT_BC);
}

/* Funzione di recupero della pallina */
int recoveryBall(int balls) {
  OnRev(OUT_A, MOVE_ARM);
  Wait(500);
  OnFwd(OUT_A, MOVE_ARM);
  Wait(500);
  Off(OUT_A);
  return (++balls);
}

/* Converte un numero in una stringa e permette di assegnare in modo esatto il tempo effettivo di spostamento tra 2 stazioni all'opportuno collegamento tra le stesse */
string arc(int num_arc){
  switch(num_arc){
    case 0: return "q";
    case 1: return "r";
    case 2: return "s";
    case 3: return "t";
    case 4: return "u";
    case 5: return "v";
    case 6: return "w";
    case 7: return "x";
    case 8: return "y";
    case 9: return "z";
  }
}
```

```
/* Stampa a monitor del Mindstorm i dati registrati */
void printTimes(int times[]){
  while(true){
    for(int i=0; i < ARRAY_LENGTH ; i++){
      ResetScreen();
      if (times[i] != 0){
        TextOut(10, LCD_LINE2, arc(i));
        NumOut(10, LCD_LINE4, times[i]);
        Wait(1000);
} } } }

/*  Muove il Mindstorm in modo preciso all'interno di una stazione di recupero */
void nextStation(int degree, int balls)
{
  moveIntoStation(STATION_RADIUS);
  rotate(degree, balls);
  moveIntoStation(5);
}

/* Funzione di controllo dello spostamento da una stazione A ad una stazione B */
int followLine()
{
 // Ingresso 2: Sensore di Luce Attivo in modalità RAW (0-1023) - LATO DESTRO Mindstorm
  // Ingresso 3: Sensore di Luce Attivo in modalità RAW (0-1023) - LATO SINISTRA
Mindstorm
 SetSensorType(IN_2, SENSOR_TYPE_LIGHT_ACTIVE);
 SetSensorMode(IN_2, SENSOR_MODE_RAW);
 SetSensorType(IN_3, SENSOR_TYPE_LIGHT_ACTIVE);
 SetSensorMode(IN_3, SENSOR_MODE_RAW);

 // Dichiarazione variabili locali
 bool loop = 1;
 short state;
 int adj_left = 0;
 int adj_right = 0;
 int timeOfWait = distanceToTime(6);
 string stateText = "AVANTI";
```

```
short move_speed_sx, move_speed_dx;
unsigned long local_start_time, local_end_time;
local_start_time = CurrentTick();
local_end_time = local_start_time;
OnFwd(OUT_BC, MOVE_SPEED);

while (loop)  {
 state = 0;
 move_speed_sx = MOVE_SPEED;
 move_speed_dx = MOVE_SPEED;
 ResetScreen();
 adj_left  = Sensor(IN_3);
 adj_right = Sensor(IN_2);
 if (adj_left > LSL_THRESHOLD && adj_right < RSL_THRESHOLD) state = 1;
 if (adj_left < LSL_THRESHOLD && adj_right > RSL_THRESHOLD) state = 2;
 if (adj_left > LSL_THRESHOLD && adj_right > RSL_THRESHOLD) state = 3;
 switch (state){
   case 1: stateText = "AGG_Sinistra";
           move_speed_sx = 0;
           break;
   case 2: stateText =  "AGG_Destra";
           move_speed_dx = 0;
           break;
   case 3: OnFwd(OUT_BC, MOVE_SPEED);
           Wait(timeOfWait);

           // Se legge entrambi i sensori sopra la soglia (bianco), riaccendiamo entrambi i motori
           //  per interrompere una possibile ROTAZIONE
           //(AGG_Sinistra o AGG_Destra), attendiamo 200ms (tempo per superare un possibile
           //  incrocio), e rileggiamo i sensori per verificare di essere correttamente in
STAZIONE
           adj_left  = Sensor(IN_3);
           adj_right = Sensor(IN_2);
           if (adj_left > LSL_THRESHOLD && adj_right > RSL_THRESHOLD){
             stateText =  "STAZIONE";
             move_speed_sx = 0;
             move_speed_dx = 0;
             local_end_time = CurrentTick();
```

```
            loop = 0;
          }
          break;
    default: stateText =  "AVANTI";
  }
  OnFwd(OUT_B, move_speed_sx);
  OnFwd(OUT_C, move_speed_dx);
 }
 Wait(500);
 ResetScreen();
 return (local_end_time-local_start_time);
}
```

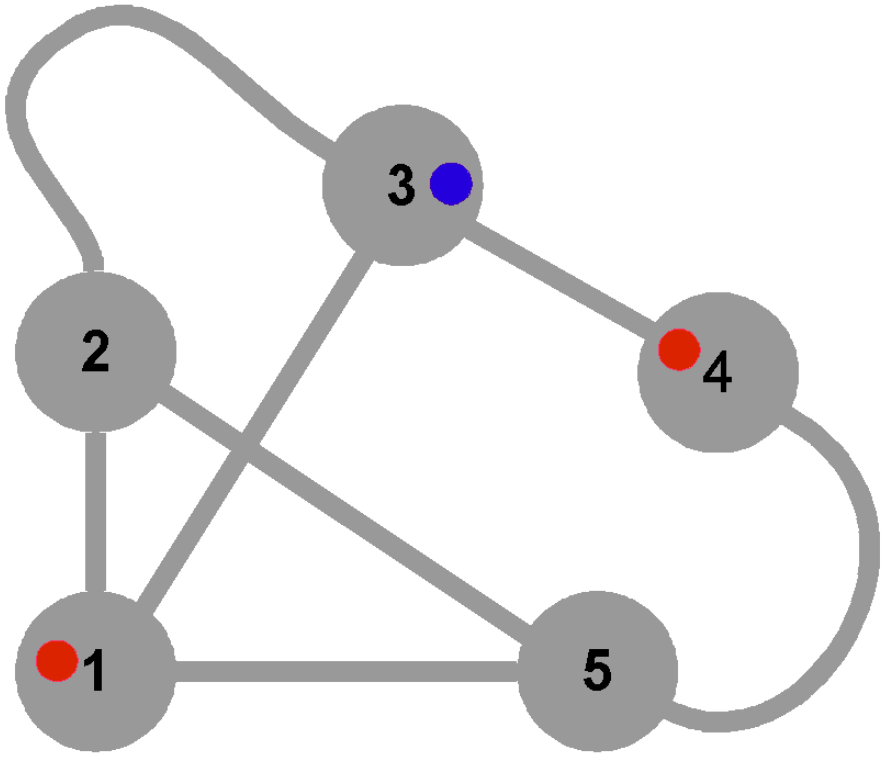
Programma principale

```
task main() {
 //inizializzazione
 int ballsNumber = 0;
 int times[];
 ArrayInit(times, 0, ARRAY_LENGTH);
 followLine();
 Inizio codice del piano (esempio)
ballsNumber = recoveryBall(ballsNumber);
nextStation(30, ballsNumber);
times[1] = followLine();
ballsNumber = recoveryBall(ballsNumber);
nextStation(-90, ballsNumber);
times[4] = followLine();
ballsNumber = recoveryBall(ballsNumber);
nextStation(-60, ballsNumber);
times[6] = followLine();
ballsNumber = recoveryBall(ballsNumber);
nextStation(0, ballsNumber);
times[9] = followLine();
ballsNumber = recoveryBall(ballsNumber);
  Fine codice del piano
 printTimes(times);
}
```

Modulo 4: l'ambiente

L'ambiente è l'area cittadina di cui il pianificatore ha una rappresentazione formale e con cui il robot interagisce in modo diretto.

Tale ambiente è stato modellato, come già visto per il pianificatore, con un grafo.



Un grafo <S,A>, in cui l'insieme dei nodi S rappresenta tutte le locazioni di recupero dei rifiuti e l'insieme A degli archi i collegamenti tra le varie locazioni.

Ogni nodo del grafo, inoltre, può avere un contenitore di rifiuti da recuperare.

Per la realizzazione del grafo si è scelto di utilizzare fogli di carta da disegno dura, di colore bianco, di formato A3. Per i contenitori dei rifiuti, invece, semplici palline di plastica.

Le scelte sono state motivate da fattori di varia natura:

- Fattori meccanici: nodi ed archi devono resistere all'abrasione causata dal passaggio del robot e dall'uso frequente di scotch nelle fasi iniziali di realizzazione del progetto
- Fattori cromatici: nodi ed archi devono essere di un colore tale da generare un elevato contrasto con la pavimentazione
- Fattori economici:

1. nelle nostre abitazioni erano disponibili alcuni fogli di carta da disegno;
2. le palline di plastica sono offerte in dotazione con i Lego Mindstorm.

La costruzione di nodi ed archi è stata guidata dal principio di economicità: per evitare qualsiasi spreco, è stato ottimizzato l'uso della carta e le posizioni di intaglio attraverso l'uso di una strumentazione adeguata:

- Decametro e calibro
- Matite
- Forbici e cutter.

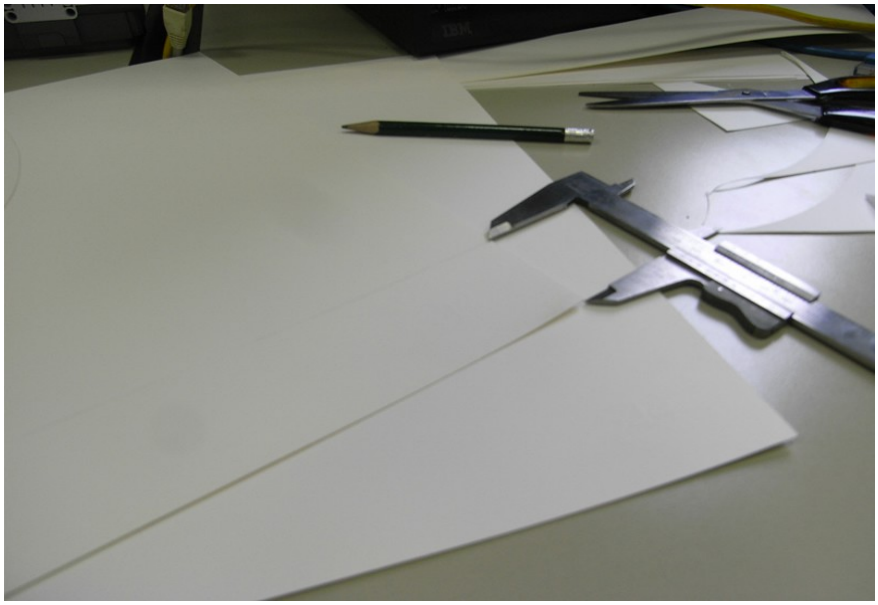


Illustrazione 9: Preparazione degli archi

Il risultato è stato quello di ottenere da ogni foglio A3 un nodo e 6 archi.

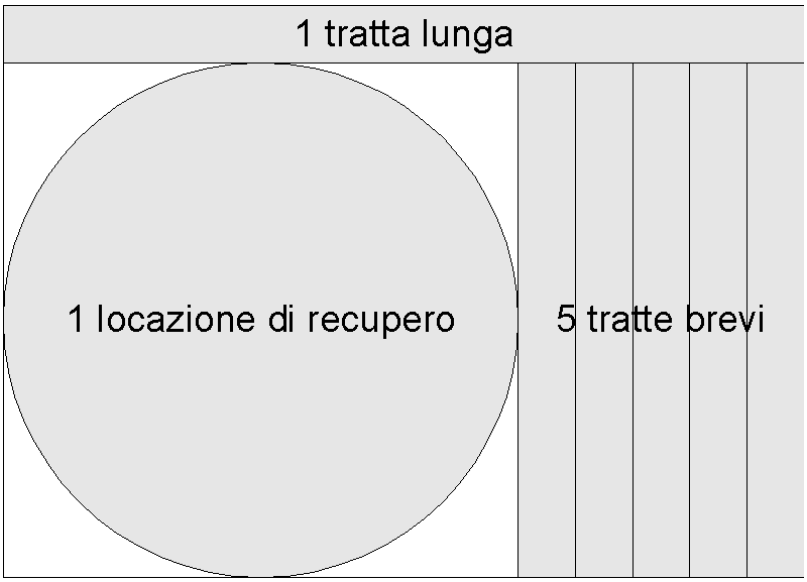


Illustrazione 10: Utilizzo dei fogli A3 per creare archi e stazioni

Cronologia di progetto

Di seguito si riporta un breve prospetto del tempo impiegato per svolgere le principali attività legate al progetto

Mese	Attività	Ore lavoro
Maggio 2007	Ideazione del progetto	39
Giugno 2007	Ideazione del robot	8
	Formalizzazione del problema in lparse	36
Luglio 2007	Formalizzazione del problema in lparse	29
Settembre 2007	Upgrade robot Smodels Applicazione php	25
Ottobre 2007	Progettazione sequenza azioni	40
Novembre 2007	Test fattibilità in Urbi e in NBC	38
Dicembre 2007	Programmazione NXT, upgrade robot, testing	52
Gennaio 2008	Programmazione NXT, upgrade robot, testing	32
	Acquisto materiale per stazioni, realizzazione Stazioni	4
Febbraio 2008	Sviluppo in NXT, testing	16
Marzo 2008	Realizzazione percorso, upgrade NXT	8

Riferimenti

- Sito ufficiale: http://www.aracni.de/spazzinoproject/

Conclusione

Spazzino Project, dopo l'analisi affrontata, si può con certezza definire il pianificatore automatico che risolve e simula il recupero dei rifiuti di un'area cittadina attraverso l'attuazione di un piano conformante eseguito da un Lego Mindstorm costruito ad hoc.

Inoltre, grazie all'interazione tra i suoi moduli principali (il pianificatore e il robot), garantisce l'ottimizzazione del tempo totale di uscita del mezzo che copre l'area cittadina. Ogni esecuzione del piano da parte del robot consente infatti, come visto, il ricalcolo del piano stesso con dati informativi che consentono di raggiungere una copertura ottimale dell'area cittadina.

Infine, Spazzino Project, come si analizzerà meglio nel prossimo paragrafo, lascia margini di miglioramento alla soluzione del problema di pianificazione affrontato data la sua efficiente modularità.

Sviluppi futuri

Spazzino Project può essere migliorato sotto 2 aspetti:

- introducendo nel modello di rappresentazione del problema azioni di sensing che regolino la capienza del contenitore di rifiuti del robot (nel modello attuale è illimitata) e la capacità del robot stesso di cambiare il percorso di recupero dei rifiuti nel caso in cui uno o più collegamenti tra locazioni sia impercorribile.

  Il modello di soluzione del problema a questo punto non sarebbe più un piano conformante ma bensì un piano condizionale.

  Esempio

  Per la soluzione della capienza del contenitore bisogna introdurre nel modello l'azione di peso dei rifiuti R, action(weigh(I,R)), per consentire al robot di verificare se sono caricabili o meno. Tale azione oltre alle causalità da considerare, che qui non trattiamo, richiede l'introduzione del costrutto *determines(weigh(I,R), loadable(I,R)),* sezione pianificatore-fase 3, che permette al pianificatore di sviluppare due rami del piano differenti:

  il primo, loadable(I,R) = TRUE, che modella il fatto che i rifiuti siano recuperati;

  il secondo, loadable(I,R) = FALSE, che modella il fatto che i rifiuti non siano caricabili per capacità residua insufficiente.

- sviluppando un modulo software, portabile su più piattaforme, che consenta di bypassare l'interfaccia standard del robot, per comunicare il codice di esecuzione al robot stesso direttamente dall'interfaccia.

Bibliografia

[1] Michael Gelfond, Vladimir Lifschitz. “Representing Action and Change by Logic Programs” (1993) Journal of Logic Programming

[2] Tran Cao Son, Chitta Baral, “Planning with Sensing Actions and Incomplete Information Using Logic Programming”, New Mexico and Arizona States University http://www.cs.nmsu.edu/~tson/ASPlan/Sensing/boolean.html

[3] Illka Niemelä and Patrik Simons. “Smodels – an implementation of the stable model and well-founded semantics for normal logic programs” Helsinki University of Technology, Department of Computer Science and Engineering

[4] Tommi Syrjänen. “Lparse 1.0 User's Manual” (1998-2000)

[5] The Unofficial LEGO MINDSTORMS NXT Inventor's Guide, O'Reilly

[6] The LEGO MINDSTORMS NXT Idea Book, O'Reilly

[7] Jin Sato's LEGO MINDSTORMS, O'Reilly

http://www.oreilly.com/catalog/lmstorms/

Riferimenti sul web

- Php: http://www.php.net/
- Lego Mindstorm: http://mindstorms.lego.com/Overview/NXTreme.aspx
- Bricx - nbc: http://bricxcc.sourceforge.net/nbc/
- Il blog dello Spazzino Project – http://www.aracni.de/spazzinoproject