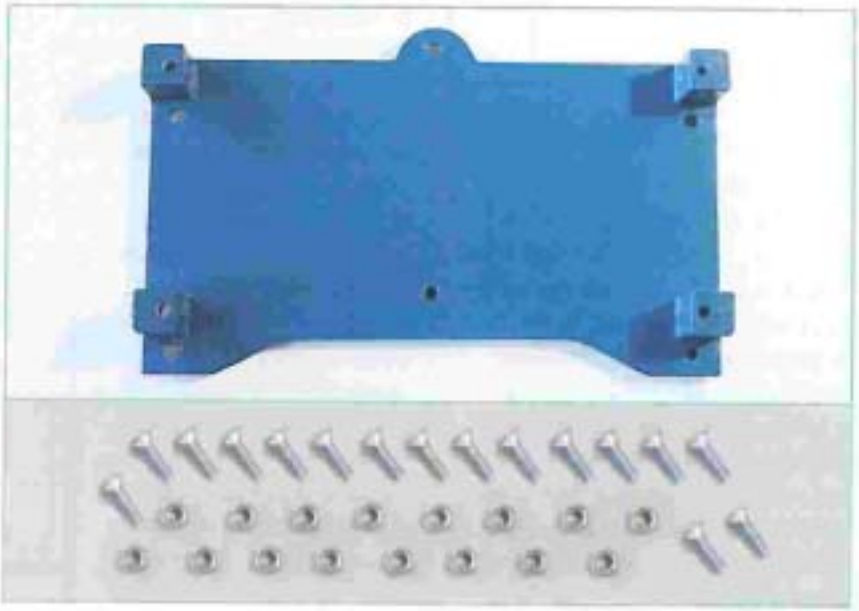
Verso l'algoritmo

Con gli allegati di questo fascicolo, si inaugura una nuova fase di implementazione hardware del robot: presto, infatti, potrai dotarlo di un nuovo supporto per la locomozione (avrai ruote più grandi e, dunque, un robot più veloce) e per l'alimentazione (grazie alle batterie ricaricabili). Rimandiamo le fasi di montaggio al momento in cui disporremo dell'intero set, completato con i prossimi due fascicoli; ora, invece, torniamo al discorso della soluzione del 'problema labirinto', iniziato a pagina 185.

STUDIO DEL PROBLEMA

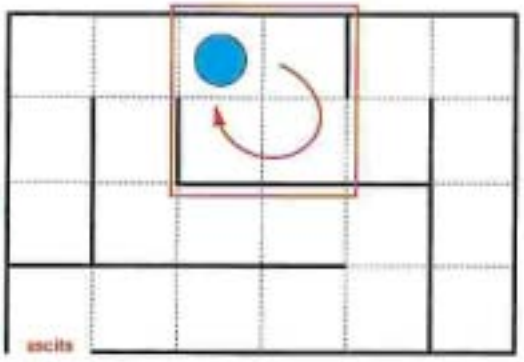
Fatte le premesse e le prime considerazioni, iniziamo lo studio vero e proprio del problema, ripartendo dalla struttura dei labirinti caratterizzata a pagina 186. Ciò che intendiamo fare è 'avvicinarci' alla stesura del programma di navigazione, tramite un algoritmo che, basato sulle regole di comportamento richieste al robot, verrà schematizzato in un diagramma di flusso (ossia una visualizzazione grafica particolare di cui si discuterà in seguito). Poiché la presenza dei baffi interessa solo l'eventuale correzione della traiettoria del robot, sarà trattata in un secondo tempo; ora, invece, ci occupiamo soltanto della navigazione basata sui rilevamenti dei sensori IR.

Suddividendo il problema in sottoproblemi è più facile schematizzarlo



Nella foto. Le viti, i dadi e il supporto plastico (per le ruote e le batterie ricaricabili) allegati a questo fascicolo.

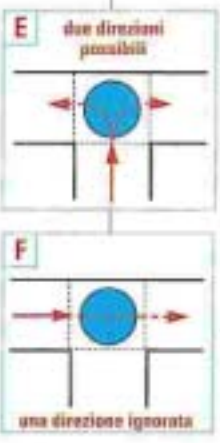
In base alle caratteristiche del nostro labirinto (in particolare a quelle delle celle in cui è stato suddiviso), possiamo anzitutto distinguere due casi particolari e ipotizzare quindi un relativo comportamento utile del robot. **(a)** Se il robot sta avanzando in una certa direzione e non rileva nessun ostacolo avanti a sé, è ragionevole lasciarlo avanzare. **(b)** Se il robot si trova di fronte a un ostacolo (che nel nostro labirinto non può che essere un muro), può trovarsi in quattro diversi tipi di celle (visti a pag. 186, nelle figure **B**, **D** e **C**: in quest'ultima, soltanto le due curve rientrano nell'ipotesi). Dunque, prima di decidere che politica di navigazione adottare, bisogna accertare il tipo di cella. Per farlo è sufficiente che il robot ruoti di 90° (a destra o a sinistra, è per ora ininfluente) e che testi con i quattro IR la presenza di ostacoli ai lati della cella. Quindi, si potrà decidere che: **(b1)** se l'unica direzione possibile è quella da cui il robot è venuto (fig. **B** a pagina 186), esso deve tornare indietro; **(b2)** se oltre a quella di provenienza esiste solo un'altra direzione possibile (fig. **C** a pag. 186), il robot deve prendere quella; **(b3)** se, infine, esistono altre due direzioni possibili (fig. **D** a pag. 186), allora deve scegliere tra loro. È facile intuire che quest'ultima situazione è quella più complessa da trattare.



POLITICHE DIVERSE

In generale, ci sono diverse politiche di scelta possibili. Ad esempio, per quel che riguarda la scelta di una direzione su tre possibili (due nuove più quella di provenienza) si può optare per l'approccio random, cioè a caso, oppure impostare una direzione preferenziale di scelta (ad esempio con una costante). Con una scelta di tipo random, la navigazione risultante dalle ipotesi (a) e (b) rappresenta un esempio di navigazione 'ibrida': casuale, ma in un ambiente dotato di caratteristiche molto ben definite, la cui conoscenza delimita e riduce parte della casualità iniziale. Sebbene non sia facile decidere a priori se sia meglio adottare una politica di scelta casuale o predefinita, è però possibile, scartare l'idea di adottarne una 'predefinita e costante', cioè molto poco flessibile. Vediamo perché. Analizzando il nostro labirinto d'esempio (riprodotto due volte in questa pagina), possiamo individuarne alcuni punti critici (riquadrati in rosso). Per esempio, se il robot si trova nella posizione illustrata qui sopra, e dovesse costantemente mantenere la destra come direzione di svolta predefinita, incapperebbe in un ciclo (indicato dalla freccia) da cui non sarebbe più in grado di uscire. In questo caso, quindi, la scelta di imboccare una direzione a caso di fronte a un ostacolo è sicuramente migliore.

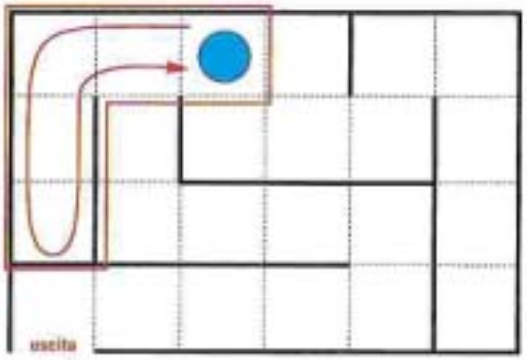
Si potrebbe tuttavia preferire una politica intermedia come quella, ad esempio, di settare sì una direzione di svolta predefinita, ma associandola a una variabile di tipo bit. In tal caso il suo valore potrà essere invertito ogni qualvolta si assecondasse la direzione predefinita: se questa fosse la destra, dopo una prima svolta a destra, se ne avrebbe una a sinistra, evitando così, ad esempio, il ciclo visto in precedenza. **Sarà dunque utile mettere in conto, oltre alle routine di navigazione che implementeremo nell'algoritmo, anche una routine che si occupi di settare una variabile relativa alla direzione di svolta predefinita.** La scelta del tipo di politica da adottare (random, cambiando direzione di svolta ogni *n* volte ecc.), invece, è per ora ininfluente; se però si avrà l'accortezza di prevedere il settaggio della direzione predefinita in una routine separata dalle altre, si potrà in seguito intervenire facilmente sull'algoritmo per sperimentare diverse politiche e, sulla base di test pratici, scegliere infine quella più adatta al caso specifico.



A sinistra e sotto. Due esempi (riquadrati in rosso) di punti critici del nostro labirinto, nel caso in cui valessero solo le ipotesi (a) e (b) premesse nel testo.

MIGLIORI GARANZIE

C'è un altro problema (illustrato in figura **F**) che riguarda le celle in cui il robot può scegliere fra tre diverse direzioni (di cui una è quella da cui proviene): un conto, infatti, è trovarsi nel caso illustrato in figura **E**, che è analogo a quello discusso prima:



di fronte all'ostacolo, il robot verifica due possibili direzioni; dovrà quindi sceglierne una, in base a una certa politica. Apparentemente simile, il caso illustrato in figura **F** solleva invece un problema diverso. Poiché il robot non incontra alcun ostacolo, in base all'ipotesi (a), andrà dritto; questo però comporta che ignori totalmente la possibilità di verificare la presenza o meno di un'ulteriore nuova direzione. Sebbene questa politica (per cui il robot, se non incontra ostacoli, non 'guarda intorno') possa essere utile (nonché snella, in termini di tempo), alcune volte impedisce al robot di uscire dal labirinto, 'incastrandolo' in un ciclo (sopra) inesorabile come il precedente.

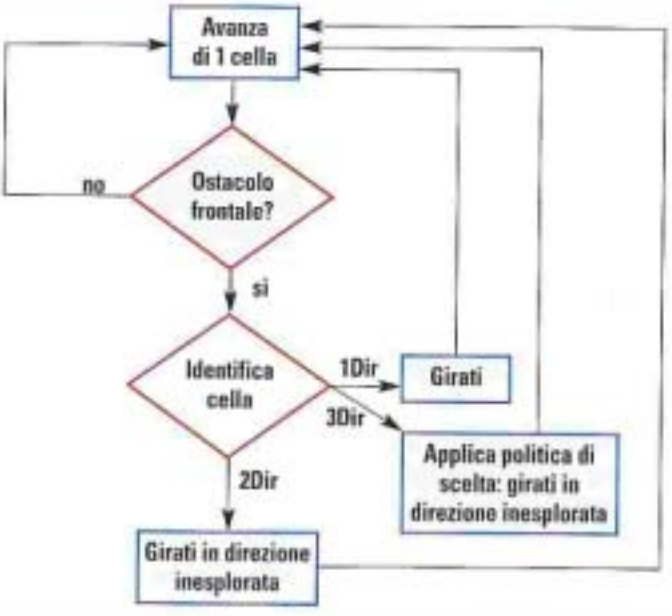
A questo punto, appare evidente che anche l'ipotesi di lavoro (a) deve essere ritoccata, in modo che, almeno in condizioni generali, non precluda a priori al robot la possibilità di vagliare tutte le strade del nostro labirinto. Una prima politica correttiva potrebbe essere quella di controllare sempre, anche in assenza di ostacoli, l'esistenza o meno di nuove direzioni, ottenendo, di fatto, una generalizzazione del discorso iniziale. Rimarrebbe però il problema della lentezza di navigazione: non solo gli incroci, ma anche ogni cella di mero transito richiederebbe infatti un test sistematico. Ancora una volta, può essere utile optare invece per una politica di scelta intermedia che, almeno in parte, velocizzi la navigazione. Innanzitutto, si può tener conto di quanto già acquisito: in prima approssimazione, infatti, non è sbagliato ignorare, in assenza di ostacoli, l'eventuale presenza di strade laterali. D'altronde, se le si ignora tanto all'andata quanto al ritorno da un vicolo cieco, esse resteranno sconosciute per sempre, costringendo il robot a tornare sui propri passi. **Un'idea quindi può essere quella di settare una variabile di stato che imponga di cercare strade alternative solo dopo che il robot sia stato costretto a tornare indietro** (come illustrato in figura G).

Nello schema. **Esempio di una prima stesura del diagramma di flusso dell'algoritmo per la navigazione con i sensori IR, discusso nel testo. Le caselle in colore verranno poi espanse.**

Fatto questo, resta comunque da decidere che cosa fare una volta scoperta una strada alternativa. Decidere a priori di imboccarla in ogni caso, infatti, può non essere sempre la scelta migliore: nella situazione illustrata in figura H, ad esempio, torneremmo da capo, come in figura F.
In generale, allora, vale la pena ribadire che in assenza di vincoli sul tempo, la politica random è preferibile perché non esclude a priori alcuna possibilità e, quindi, garantisce al robot di trovare l'uscita dal labirinto. Tipicamente però, i vincoli di tempo esistono (basti pensare a certe gare): quindi, anche a costo di perdere alcune garanzie di successo, diventa necessario studiare politiche più efficienti, spesso intermedie tra il caso e l'assoluta predeterminazione.

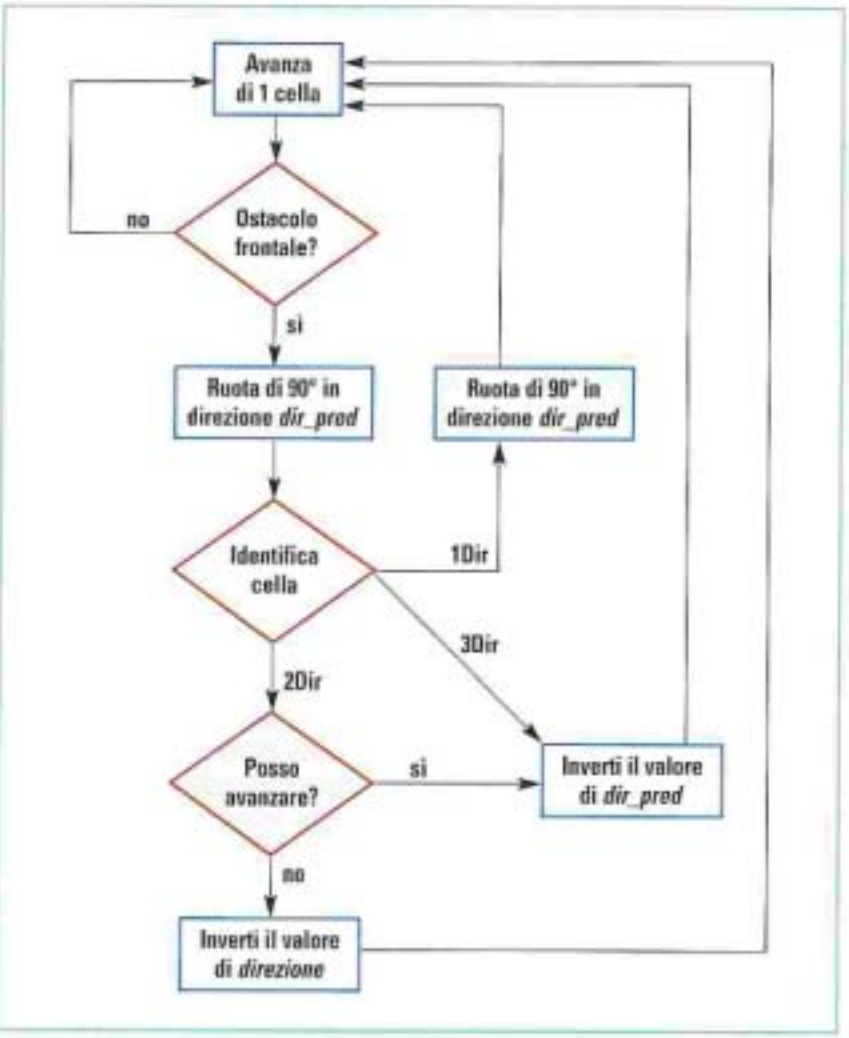
PRIMA STESURA DELL'ALGORITMO

Arrivati a questo punto, si può iniziare a scrivere l'algoritmo di navigazione vero e proprio, di cui stenderemo ora le linee essenziali, considerando solo gli IR: toccherà poi al singolo progettista perfezionarlo e ampliarlo. **La fase di stesura dell'algoritmo è molto delicata e richiede perciò grande attenzione.** In seguito, infatti, avendo di fronte l'algoritmo ben schematizzato (piuttosto che il codice di un programma), sarà molto più facile scovare eventuali errori concettuali; inoltre, è da un algoritmo ben scritto che si ricava, di solito, un codice ben strutturato e, quindi, facilmente migliorabile. **Lo strumento grafico tipicamente utilizzato per visualizzare un algoritmo è il cosiddetto 'diagramma di flusso' (sotto): un particolare schema costituito da caselle correlate da frecce.**



La sequenza di caselle che si incontrano seguendo le frecce è il 'flusso di un algoritmo'. Ogni casella contiene un testo cui corrisponde uno stato e un'azione: quest'ultima dipende dalla forma della casella, rettangolare o romboidale. Per convenzione, queste ultime (in rosso nei diagrammi di queste pagine) rappresentano un test e, dunque, una diramazione del flusso dell'algoritmo (tipicamente un bivio sì/no, ma anche un trivio, come 1Dir/2Dir/3Dir del diagramma a destra, un quadrivio ecc.). Alla luce delle ipotesi (a) e (b) discusse e ritoccate nelle pagine precedenti, possiamo iniziare a stendere il diagramma di pag. 191, che presenta due test: il primo verifica la presenza di un ostacolo frontale, il secondo esegue un rilevamento sulle pareti laterali. Questo secondo test, inoltre, include un'altra operazione fondamentale: ossia la rotazione (di 90°) necessaria perché gli IR (frontali e posteriori) rilevino ostacoli che siano laterali. Ricordiamo inoltre che tutti e quattro i sensori IR memorizzeranno l'esito del proprio rilevamento in un'unica variabile di stato a 4 bit, gestibile da un branch (di cui si è già ampiamente discusso in sede di programmazione); a un livello di astrazione più elevato (cioè quello del diagramma di flusso), basterà semplicemente tradurre tale variabile nell'informazione (sì/no) relativa alla presenza o assenza di un ostacolo.

Il diagramma di flusso può essere espanso in più fasi successive



GESTIRE LE POLITICHE

Consideriamo ora la porzione del diagramma relativa alla gestione della politica di scelta di una direzione. Essendo un passaggio importante, proviamo a dettagliare meglio il diagramma (come illustrato sopra), implementando in questo blocco una politica intermedia, del tipo discusso in precedenza. In pratica, si tratta di associare la direzione predefinita a un flag (variabile binaria) che venga invertito non appena una svolta ne assecondasse il valore. Aggiungiamo quindi una variabile di stato di tipo binario, che chiameremo *dir_pred*, la cui logica è la seguente: se il robot incontra un ostacolo frontale mentre avanza, ruota di 90° rispetto a *dir_pred*, testa la presenza di ostacoli laterali e, in base alla rilevazione degli IR, identifica il tipo di cella (a 1, 2 o 3 direzioni) in cui si trova. A questo punto entrano in gioco le politiche diverse. Se non ci fosse alternativa alla direzione di provenienza (1Dir nel diagramma), il robot ruoterà di altri 90°, sempre secondo *dir_pred*: giratosi, potrà tornare sui suoi passi. Nel caso il robot, che si è già girato secondo *dir_pred*, si trovasse invece in

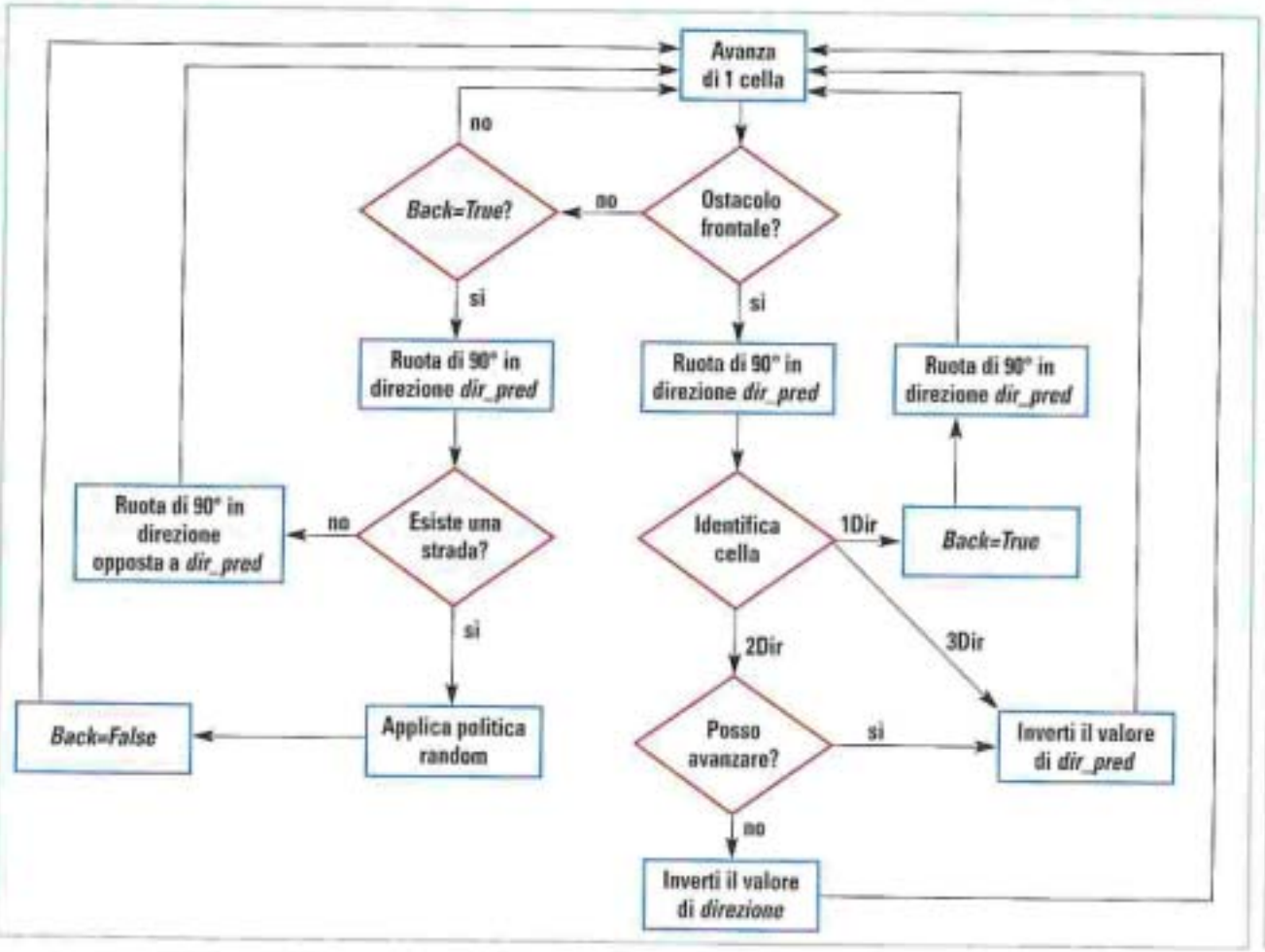
Nello schema. **Le caselle in colore rappresentano la prima espansione del diagramma, rispetto alle politiche di scelta.**

una cella a tre direzioni (3Dir), non deve far altro che avanzare: prima, però, invertirà il valore della variabile, a seconda della politica scelta (ogni volta, ogni due ecc.). Il caso di celle a due direzioni (2Dir) è invece un po' più complesso: infatti, se la direzione individuata in accordo con *dir_pred* è percorribile, il comportamento è lo stesso di 3Dir; se invece il robot, compiuta la prima rotazione secondo *dir_pred*, verificasse di avere ancora di fronte un ostacolo, invertirà la propria marcia (grazie alla variabile *direzione*, già discussa); invece *dir_pred*, non essendo stata assecondata, resta invariata.

BACK: TRUE OR FALSE?

Dettagliato così il diagramma, rimane ancora una questione da risolvere: quando occuparsi della verifica di eventuali strade laterali? In accordo con quanto stabilito, il rilevamento avverrà solo in seguito all'incontro di un vicolo cieco, ossia al test di una cella a una sola direzione. Per farlo, dovremo quindi prevedere un nuovo flag di stato che memorizzi questo evento. Tale flag, che chiameremo *Back* ('Indietro') è all'inizio disattivato. Attivato dopo un vicolo cieco (*Back=True*, ossia 'Vero', nel diagramma sotto), diventerà necessario eseguire il test a ogni passo, anche in assenza di ostacoli frontali: solo così, infatti, sarà possibile rilevare anche l'eventuale transito successivo in una casella con tre direzioni. Se l'esito del test *Back=True?* è positivo, infatti, ci si troverà proprio in quel tipo di cella e si dovrà applicare la politica più adeguata (nel diagramma, si è scelta quella random), in seguito alla quale il flag va nuovamente disattivato (*Back=False*, cioè 'Falso'). Il diagramma di flusso così ottenuto (sotto), illustra perciò uno dei possibili algoritmi risolutivi del nostro problema.

Nello schema. Ulteriore espansione del diagramma in funzione della variabile *Back* (nelle caselle in colore).



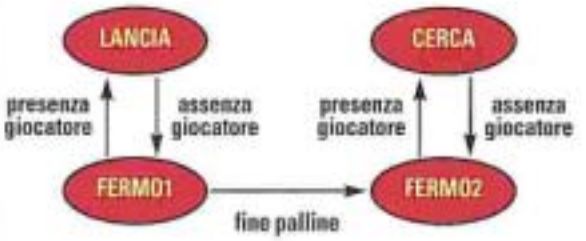
ALCUNE OSSERVAZIONI

Oltre alla variabile a 4 bit dei sensori IR, ci siamo serviti di tre flag di stato: *direzione*, che memorizza la direzione corrente; *dir_pred*, che memorizza quella di svolta predefinita; *Back* che, inizializzata a 0 (*False*), memorizza il fatto di aver incontrato una cella con una sola direzione ammissibile (un vicolo cieco). Programmato con un algoritmo come questo, il robot è di fatto assimilabile a una cosiddetta 'macchina a stati finiti' che, in funzione della variabile *Back*, prevede principalmente, due macro-stati: il 'prima' e il 'dopo' l'arrivo in un vicolo cieco del labirinto. Per quanto riguarda i sensori IR, va ricordato che l'esito del rilevamento degli ostacoli frontali dipende, di fatto, dalla sola coppia di sensori (Front o Back) che, di volta in volta, si trova a essere frontale rispetto alla direzione di marcia. Il test di eventuali strade alternative, invece, sfrutta entrambe le coppie di IR contemporaneamente. Volendo risparmiare spazio in memoria (qualora servisse per gestire altre istruzioni), per rilevare gli ostacoli frontali si potrebbe decidere di usare un solo IR di ogni coppia, rinunciando però a una maggior precisione. Riguardo ai baffi, adibiti alla sola correzione della traiettoria, sarà invece sufficiente prevedere, in ogni routine di avanzamento e rotazione (definite all'inizio), una chiamata a una subroutine che si occupi di verificare eventuali collisioni con i muri e, poi, di modificare la traiettoria.

Dal diagramma di flusso dell'algoritmo si ricaverà il programma

UNA MACCHINA A STATI FINITI

Una macchina (o automa) a stati finiti è un particolare tipo di macchina il cui comportamento in un ambiente (reale o astratto) è osservabile in un numero discreto e definito di istanti. Di conseguenza, si può distinguere anche un corrispondente numero finito di diversi stati di funzionamento della macchina stessa. Tali stati, inoltre, possono essere caratterizzati da un numero finito di variabili (dette 'di stato'). Le variabili cosiddette 'ambientali', invece, caratterizzeranno l'interazione della macchina con l'ambiente: il passaggio da uno stato all'altro, infatti, è il risultato dell'occorrenza di un evento esterno. Per capire cosa sia uno stato, bisogna considerare il funzionamento della macchina da due punti di vista molto diversi: uno interno e uno esterno alla macchina stessa. Per un osservatore interno alla macchina, un cambiamento di stato è caratterizzato dalla variazione del valore di almeno una delle variabili di stato della macchina. Date quindi *n* variabili di stato, a ogni stato corrisponde una loro precisa "combinazione" (cosiddetta 'vettore di stato'); non vale invece l'inverso, giacché i vettori possibili sono infiniti, mentre gli stati della macchina devono essere finiti. Per quel che riguarda un osservatore esterno, invece, la variazione di stato viene percepita unicamente come variazione del comportamento della macchina rispetto al ripetersi di un evento ambientale.



Fissato uno stato e un evento, infatti, la macchina risponde sempre allo stesso modo; se invece l'evento è lo stesso ma la risposta cambia, deve essere cambiato lo stato. Nella fase di progettazione, una macchina a stati finiti può essere schematizzata in un diagramma degli stati (in basso a sinistra), cioè un grafo che abbia per nodi gli stati e per archi i cambiamenti. Tipicamente, gli stati di funzionamento vengono definiti a partire dal comportamento esterno che si vuole ottenere. In seguito, poi, ci si preoccuperà di implementare questi stati con la definizione di opportune variabili di stato e di regole di comportamento. Un robot lancia-e-raccattapalle (sopra, quello dell'inglese Roke Manor Research) è un esempio di macchina a stati finiti: la variabile di stato è il numero di palline raccolte; la variabile ambientale è la presenza di un giocatore avversario. Il robot parte nello stato FERM01, con *NumeroPalline*>0. In presenza di un giocatore, lancia una palla (si porta cioè nello stato LANCIA) per poi tornare in FERM01. Quindi controlla la scorta di palle e, se *NumeroPalline*=0, si porta in FERM02. Al sopraggiungere dello stesso evento ambientale di prima (la presenza del giocatore), il robot si comporterà ora diversamente: si porterà infatti nello stato CERCA, restandoci finché avrà recuperato altre palline.